

SHARP

SHARP
COMPUTER
SOFTWARE

△▽68000用

Compiler

≡ver2.0≡

PRO-68K

ソースコードデバッガマニュアル

 68000用

COMPILER  ver2.0 
PRO-68K

ソースコードデバッガマニュアル

SHARP

はじめに

本書は、「C compiler PRO-68K ver2.0」(以下、XC コンパイラと表記します)でプログラム開発を行うときに XC のソースを参照しながら効率的なデバッグが行えるソースコードデバッグの使用方法について記述しています。

なお、別冊の「C ユーザーズマニュアル」に XC コンパイラの商品構成、動作環境の作成方法、各マニュアルの内容が説明されています。

初めて本プログラムをご使用になる方は、必ず「C ユーザーズマニュアル」を先にご覧ください。

ソースコードデバッガ (SCD) は、シンボルによりデバッグ可能なデバッガ (DB) の機能に加えて、マウス表示画面 (マウススクリーン) を利用したフルスクリーンモードを用いて、デバッグ中に C 言語のソースを参照したり、C 言語で宣言された変数を参照／変更したりできる機能を備えています。

SCD の主な機能は、次の通りです。

- スクロール可能な逆アセンブラ機能
- 32 個までのブレークポイント
- レジスタ／メモリの参照および変更
- 命令のトレース／ステップ実行
- トレース履歴表示
- ウォッチポイント (条件付きブレークポイント)
- トレースポイント (メモリ監視)
- C 言語ソースプログラムの表示
- C 変数の参照／変更

本書の構成

本書は以下の内容から構成されています。

第 1 章 準備

ソースコードデバッガ (SCD) を使用するための準備段階となる実行プログラムのコンパイル、SCD の動作環境について説明します。

第 2 章 起動

SCD の起動方法、コンソールモードをはじめとした多彩なオプション、およびコンフィギュレーションファイルの作成、オンラインヘルプについて述べます。

第 3 章 操作モード

SCD が持つ 2 つの画面モード、2 つの表示モード、およびデバッグ対象プログラムに含まれるデバッグ情報の種類について述べます。

第 4 章 コマンド

SCD のコマンド表記、C 言語の式表記、および SCD のコマンドの書式を述べるとともに、SCD のコマンドを一覧表として詳述します。

第 5 章 フルスクリーン

マウススクリーン全面を使用して、マウスやカーソルによる視覚的な操作を行うフルスクリーンモードについて、具体的な操作方法を述べながら解説します。

第 6 章 使用例

C 言語で記述され、XC コンパイラでコンパイルされたプログラムを SCD でデバッグする場合の基本的な手順、およびサンプルプログラムの説明、ソースプログラムの作成&コンパイル、プログラムの実行などについて詳述します。

付録

キー操作一覧と SCD エラーメッセージ一覧を添付します。

CONTENTS

第 1 章 準 備

1. 1	実行プログラムのコンパイル	3
1. 2	SCD の動作環境	4
1. 2. 1	最低 256kB 以上のフリーエリア	4
1. 2. 2	FLOAT2. X.....	4
1. 2. 3	デバッグ対象の実行プログラム、 およびソースプログラム	4

第 2 章 起 動

2. 1	起動方法	7
2. 2	オプション説明	8
2. 3	コンフィギュレーションファイルの作成	9
2. 4	オンラインヘルプ	10

第 3 章 操作モード

3. 1	画面モード	13
3. 2	アセンブリ表示モードとソース表示モード	14
3. 3	デバッグ対象プログラムに含まれる デバッグ情報の種類	15

第 4 章 コマンド

4. 1	SCD のコマンド表記	19
4. 1. 1	〈式〉 〈アドレス〉	19
4. 1. 2	〈サイズ〉	19
4. 1. 3	〈レンジ〉	20
4. 2	C 言語の式表記	21
4. 2. 1	〈C 評価式〉	21
4. 2. 2	〈フォーマット〉	22
4. 3	コマンドの書式	23
4. 4	コマンド一覧表	75

第5章 フルスクリーン

5.1 概要	79
5.2 プルダウン	80
5.2.1 プルダウンメニューの使い方	80
5.2.2 プルダウンメニュー一覧	81
5.3 マウス操作	82
5.3.1 リストスクリーン上での操作	82
5.3.2 スクリーンサイズの変更	82
5.4 エスケープキー	83
5.5 カーソル	84
5.5.1 コマンド投入スクリーン上のカーソル	84
5.5.2 リストスクリーン上のカーソル	84
5.6 コンソール	85
5.7 プルダウンメニューのコマンド	86

第6章 使用例

6.1 SCD の実際の使用例	97
6.2 サンプルプログラムの説明	100
6.3 ソースプログラムの作成	102
6.4 ソースプログラムのコンパイル	103
6.5 プログラムの実行	105
6.6 SCD の起動とプログラムのデバッグ	107
6.7 ソースプログラムの修正・再コンパイル・再実行	112

付 録

キー操作一覧	117
SCD エラーメッセージ一覧	119

索 引

50 音順	121
アルファベット順	124

第1章

準備

実行プログラムのコンパイル
SCD の動作環境

本章では、ソースコードデバッガ (SCD) を使用するにあたって、準備段階として必要となる実行プログラムのコンパイル、および SCD の動作環境について説明します。

1.1

実行プログラムのコンパイル

SCD を使用するためには、まずデバッグ対象のプログラムをコンパイルしなければなりません。

バージョン 2.00 以降のコンパイラを用意してください。

実行ファイルに拡張シンボル情報を付加するオプション (/Ns) を付けてコンパイルしてください。

なお、コンパイル時にオブティマイズを指定すると拡張シンボル情報の出力が無効になりますので注意してください。

1.2 SCDの動作環境

以下は SCD を動作させるための環境に必要な条件です。

1.2.1 最低256kB 以上のフリーエリア

SCD 自身のメモリ領域、およびチャイルドプロセスのための領域で最低でも 256kB 以上の領域が必要です。

このほかにデバッグ対象プログラムが動作する領域と、シンボル情報、拡張シンボル情報、ソースコードをロードする領域が必要になります。

デバッグ対象プログラムのサイズ、ソースコードのサイズが大きくなるにしたがい、さらに大きなフリーエリアが必要になります。

1.2.2 FLOAT2. X

SCD は C 言語の式を評価するために浮動小数点演算ドライバを必要とします。

一般的に C コンパイラまたは C でコンパイルされたプログラムの実行時には FLOAT2. X (FLOAT3. X でも可) がメモリに常駐していることが必要条件となります。

1.2.3 デバッグ対象の実行プログラム、およびソースプログラム

XC コンパイラバージョン 2.00 以降によって、拡張シンボル情報付きでコンパイルされた実行プログラム、およびソースプログラムをカレントディレクトリに用意してください。

第2章

起 動

起動方法

オプション説明

コンフィギュレーションファイルの作成

オンラインヘルプ

本書では、SCD の起動方法、コンソールモードをはじめとした多彩なオプション、およびコンフィギュレーションファイルの作成、オンラインヘルプについて説明します。

2.1 起動方法

コマンドラインから次の書式で起動してください。

なお、SCD は画面モードが 768×512 ドットモードで動作します。

SCD [<オプション>] [<実行ファイル名>] [<コマンドライン>]

オプション

以下のいずれか、または組み合わせを指定します。

SCD に対するオプションは、実行ファイル名の前に指定しなければなりません。

/t	コンソールモード
/r	リモートコンソールモード
/p<パレットコード>	ペンカラー指定
/b<パレットコード>	背景カラー指定
/c<メモリサイズ>	チャイルドプロセスに与えるメモリ

実行ファイル名

デバッグ対象の実行ファイル名を拡張子、x を付けて指定します。

実行ファイル名は SCD を起動したあとで R コマンドにより指定することも可能です。

コマンドライン

実行ファイルに与えるコマンド列があれば、それを指定します。

コマンドラインは、SCD を起動したあとで C コマンドにより指定することも可能です。

2.2 オプション説明

/t コンソールモード

SCD の画面モードをコンソールモードにします。

このオプションを指定しない場合は、フルスクリーンモードになります。

/r リモートコンソールモード

RS-232C 回線を通した端末側から、SCD を操作するモードにします。

このモードでの操作は、コンソールモードと同等です。

／p<パレットコード> ペンカラー指定

フルスクリーンモードでの文字色を指定します。

パレットコードは、緑、赤、青の順に、0～fまでの16段階で指定します。

例 /p000 ペンカラー = 黒

／p オプションを省略すると、システム既定値のままで起動します。

/b<パレットコード> 背景カラー指定

フルスクリーンモードでの背景色を指定します。

パレットコードは、緑、赤、青の順に、0～fまでの16段階で指定します。

例 /bfff 背景カラー = 白

／b オプションを省略すると、システム既定値のままで起動します。

／c<メモリサイズ> チャイルドプロセスに与える
メモリサイズ

チャイルドプロセス（シェル呼び出し）、およびソースファイル読み込みのバッファに使用するメモリサイズをkB単位で指定します。

省略すると 128kB が確保されます。

例 /c256

2.3 コンフィギュレーションファイルの作成

SCD のオプション、実行ファイル名、コマンドラインは、コンフィギュレーションファイルによっても指定することができます。

SCD. X の存在するディレクトリに SCD. CNF という名前のファイル（コンフィギュレーションファイル）が存在している場合は、まずそのファイルが読み込まれて、そのなかに書かれてあるオプション等が設定されます。コンフィギュレーションファイルは、テキストエディタで作成してください。

例 <SCD. CNF の内容（一例）>

/p000 /bfff /c256

（文字色＝黒、背景色＝白、チャイルドプロセスのメモリ予約＝256kB）

2.4 オンラインヘルプ

SCD 動作中に [HELP] キー、または、H コマンドで参照できるヘルプファイル "SCD. HLP" が付属しています。

SCD. X と同じディレクトリに入れておきます。

ヘルプファイルの形式は、通常のテキストファイル形式です。

第3章

操作モード

画面モード

アセンブリ表示モードとソース表示モード

デバッグ対象プログラムに含まれるデバッグ情報の種類

本章では、SCD が持つ 2 つの画面モード（フルスクリーンモード、コンソールモード）と 2 つの表示モード（アセンブリ表示モード、ソース表示モード）、およびデバッグ対象プログラムに含まれる、デバッグ情報の種類について説明します。

3.1 画面モード

SCD は 2 つの画面モードを持っています。

フルスクリーンモード

マウススクリーン全面を使用したモード。

デバッグ対象プログラムの表示出力に影響を与えることなく、デバッグを行なうことができます。

また、マウスを併用することができ、視覚的・直接的に操作することができます。

コンソールモード

コンソール入出力のみによるモード (DB と等価な操作)。

操作性はフルスクリーンモードよりも劣りますが、デバッガとしての機能はフルスクリーンモードとほぼ同等です。

このモードの場合は、RS-232C 回線を通じて他の端末から操作することも可能です。

アセンブリ表示モード

アセンブリ表示モードは、1行が機械語1命令に対応しており、これが実行の最小単位になります。

ソース表示モード

このモードは、1行がC言語の1行に対応しており、これが実行の最小単位になります。

ソース表示モードは、拡張シンボル情報付きの実行ファイルのデバッグ時のみに利用できます。

より詳細なアセンブリレベルでのデバッグを行なう必要があれば、アセンブリ表示モードに切り替えることも可能です。

その場合でも、逆アセンブル表示の中にソース行を混在させて表示可能です。

シンボル情報なし

リンカの/x(シンボルテーブルの出力禁止)オプションなどにより、デバッグの為のシンボル情報を含まない形の実行ファイルがこれに該当します。オブジェクトの参照は、すべて絶対番地で行わなければなりませんので、最もデバッグしにくい形式です。

シンボル情報あり

実行プログラムのファイルにシンボル情報が付加されたもので、シンボル名によりオブジェクトを参照、表示することが可能です。

通常はアセンブラレベルで外部宣言されたラベルをシンボルとして参照できます。

シンボル情報の参照は、SCD および DB のいずれでもサポートされています。

拡張シンボル情報あり

XC コンパイラバージョン 2.00 以降には、拡張シンボル情報を出力する機能 (/Ns オプション) が備わっています。

これを利用すると C のソース、変数名などを参照しながらデバッグすることが可能になります。

SCD を使用する最大のメリットは、C ソースを参照しながらのデバッグができる点にあります。

第4章

コマンド

SCD のコマンド表記

C 言語の式表記

コマンドの書式

コマンド一覧表

本章では、SCD のコマンド表記、C 言語の式表記および SCD のコマンド書式について解説します。

また、SCD のコマンドを一覧表として掲載しています。

4.1 SCDのコマンド表記

4.1.1 <式> <アドレス>

式、アドレスは、アセンブラレベルでのアドレスを指定する式を記述できます。

使用できる演算子

+	加算
-	減算
*	乗算
/	除算
%	剰余
&	論理積
	論理和
!	論理否定
^	排他的論理和

使用できるアドレス表現

03c3	(16 進)
¥123	(10 進)
_1100011	(2 進)
.symbol	(アセンブラのシンボル名)
.23	(ソース行番号によるアドレス指定)
..filename. 12	(..ファイル名, 行番号によるアドレス指定)

拡張子は省略します。

4.1.2 <サイズ>

サイズ：S (バイト) W (ワード) L (ロングワード)

サイズを省略すると W (ワード) とみなします。

4.1.3 <レンジ>

レンジ：<開始アドレス> <終了アドレス>

または

<開始アドレス> L<長さ (バイト数)>

4.2 C言語の式表記

4.2.1 <C 評価式>

C 言語の式の記述と同じ形式で記述します。

例	123	(10 進)
	0x6a000	(16 進)
	i	(変数 i)
	* p	(ポインタ p の指しているメモリー内容)

変数のスコープ（可視範囲）は、現在トレース中のプログラムカウンタに依存します。

すなわち、優先度の高いほうから記すと、

- 現在ブロック内で宣言された変数
- 現在関数内で宣言された変数
- 現在モジュール内で宣言された静的変数
- グローバルに宣言された変数

の順で検索されます。

現在のプログラムカウンタがどの関数にも属さない場合や、関数のエントリーの先頭などでフレームポインタが設定されていない場合は、グローバル変数のみが参照可能です。

C 言語で表現できる型、演算子は、ほとんどサポートされていますが、例外として、3 項演算子（? : ）は使用できません。

キャスト演算子については、単純なキャスト（[int *] のように基本的な型名と 2 つまでの間接演算子を用いる程度）のみが使用できます。

変数の値を変更したい場合は、代入式を記述してください（変数名 = 値）。

それ以外の場合でも、式のなかに副作用を起こす演算子（代入、インクリメント、デクリメント、関数呼び出しなど）を記述した場合は、実際に副作用を起こすことになります。

4.2.2 <フォーマット>

フォーマットは、C 言語の式を表示する書式を次のいずれかの文字で指定します。

s : 文字列
c : 文字
x : 16 進数

フォーマットを省略すると、式の値は通常 10 進数で表示されます。

例 —argv[0] の値を文字列フォーマットで表示する
 —? argv[0] ; s

4.3 コマンドの書式

SCD のコマンドの書式は次の通りです。

<コマンド名> [**<パラメータ>**]

コマンド名

アルファベット、または記号の1文字ないし2文字で表したコマンド名を指定します。

パラメータ

コマンドが受け取るパラメータを指定します。

パラメータの種類や数は、各コマンドごとに異なっています。

一般的には、そのコマンドの使用する値、アドレスを表す数値、あるいは式、シンボル名、C言語の変数名あるいは式、ファイル名と行番号によるオブジェクトアドレス指定などがあります。

その他

DBでは、コマンドをコロン(:)で区切って、1行に複数のコマンドが記述できたのですが、SCDでは、必ず1行にひとつのコマンドを記述してください。

それでは、次ページから各コマンドについて説明します。

具体的な使用例については、本書第6章にて解説します。

また、DBと重複するコマンドについては、アセンブルマニュアル「6.4 デバッガのコマンド」も参照してください。

Assemble

書式 A [<アドレス>]
AN [<アドレス>]

機能 アセンブル

解説 MPU68000 のニーモニックをアセンブルしてメモリに格納します。
<アドレス>は、アセンブル開始アドレスを指定します。
A コマンドでは、現在メモリに入っている命令のニーモニックが表示された後にニーモニックの入力になります。
入力モードから抜けるには、ピリオド (.) を入力して、リターンキー^④を押してください。
AN コマンドは、A コマンドと同様ですが、現在メモリに入っている命令のニーモニックは表示されません。
リターンキー^④のみで、SCD のコマンド待ちに戻ります。

例

```
-A④  
--main  
000BA52E      lea      $000BD138,A7  
               .④  
-A .17④  
17:           int c = 0, i = 0;  
000BA344      clr.l   $FFFC(A6)  
               .④  
-A . _main④  
_main  
000BA340      link    A6, # $FFF8  
               .④  
-■
```


Display Breakpoint

書 式 B

機 能 ブレークポイントの表示

解 説 B (Set Breakpoint) コマンドで作成した全ブレークポイントの現在の情報を表示します。

このとき、表示される情報は以下の通りです。

- (1) ブレークポイント番号 (00~1f)
- (2) 有効 (e) / 無効 (d) の別
- (3) ブレークポイントアドレス
- (4) ブレークカウント
- (5) ブレークカウント設定値

ブレークポイントが設定されていないと、“no break pointer” と表示されます。

例

```
-B
no break pointer
-B . _main
-B
00 e 000BA340(0000;0001)      . _main
-B . 17                        ←17行にブレークポイントを設定
-B
00 e 000BA344(0000;0001)
-■
```

Set Breakpoint

書 式 B[<ブレークポイント番号>] <アドレス> [<カウント>]

機 能 ブレークポイントの設定

このブレークポイントは、G コマンドで作成するブレークポイントと違って、BC コマンドを使用しない限り、削除されません。
ブレークポイントは、32 箇所まで設定することができます。

解 説

指定した<アドレス>にブレークポイントを設定します。

プログラム実行 (G コマンド) 時にブレークポイントに達すると、プログラムは停止し、SCD のコマンド入力待ちとなります。

<ブレークポイント番号>は、設定するブレークポイントの番号 (0~1f) を指定します。

B コマンドと<ブレークポイント番号>の間に空白を入れてはいけません。

また、<ブレークポイント番号>が a~f のときは、0d~0f のように、1 文字目に数字の 0 を付けてください。

もし、番号に c、d、e を指定しようとして、Bc、Bd、Be と入力した場合、BC、BD、BE コマンドとして解釈されてしまいますので注意してください。

<ブレークポイント番号>を省略すると、自動的に空いているブレークポイント番号を割り当てます。

<アドレス>は、ブレークしたい命令語の番地を指定します。

<カウント>は、ブレークするまでに通過できる回数 (ブレークカウント設定値) を指定します。

<カウント>を省略すると、1 とみなします。

例

```
-B  
no break pointer  
-B .17  
-B  
00 e 000BA344(0000;0001)  
-■
```

←17行にブレークポイントを設定

Clear Breakpoint

書 式 BC <ブレークポイント番号>

機 能 ブレークポイントの削除

解 説 設定されているブレークポイントを削除します。
 <ブレークポイント番号>には、ブレークポイント番号 (0~1f) を指定します。
 ブレークポイント番号は、複数個指定できます。
 <ブレークポイント番号>にアスタリスク(*)を指定すると、全ブレークポイントを削除します。

例

```
-B
00 e 000BA344(0000;0001)
01 e 000BA34C(0000;0001)
-BC0                                ←ブレークポイントNo0の削除
-B
01 e 000BA34C(0000;0001)
-BC*                                ←全ブレークポイントNoの削除
-B
no break pointer
-■
```


Disable Breakpoint

書 式 BD <ブレークポイント番号>

機 能 ブレークポイントの無効化

解 説 ブレークポイントを一時的に無効化します。
ブレークポイントは、削除されるわけではないので、EB コマンドにより、いつでも有効化できます。
<ブレークポイント番号>には、無効化したいブレークポイント番号 (0~1f) を指定します。
ブレークポイント番号は、複数個指定できます。
<ブレークポイント番号>にアスタリスク(*)を指定すると、全ブレークポイントを無効化します。

例

```
-B [F]
00 e 000BA344(0000;0001)
01 e 000BA34C(0000;0001)
-BD0 [F]                                ←ブレークポイントNo0の無効化
-B [F]
00 d 000BA344(0000;0001)
01 e 000BA34C(0000;0001)
-BD* [F]                                ←全ブレークポイントNoの無効化
-B [F]
00 d 000BA344(0000;0001)
01 d 000BA34C(0000;0001)
-■
```


Enable Breakpoint

書 式 BE <ブレークポイント番号>

機 能 ブレークポイントの有効化

解 説 BD コマンドにより、一時的に無効化されたブレークポイントを有効化します。
 <ブレークポイント番号>には、有効化したいブレークポイント番号 (0~1f) を指定します。
 ブレークポイント番号は、複数個指定できます。
 <ブレークポイント番号>にアスタリスク(*)を指定すると、全ブレークポイントを有効化します。

例

```
-B 0
00 d 000BA344(0000;0001)
01 d 000BA34C(0000;0001)
-BE0                                ←ブレークポイントNo0の有効化
-B 0
00 e 000BA344(0000;0001)
01 d 000BA34C(0000;0001)
-BE*                                ←全ブレークポイントNoの有効化
-B 0
00 e 000BA344(0000;0001)
01 e 000BA34C(0000;0001)
-■
```

Break count Reset

書 式 BR

機 能 ブレークカウン트의初期化

解 説 全ブレークポイントの通過回数カウンタをクリアします。

例

```
-B [F]
00 e 000BA41E(0007;0001)
01 e 000BA432(0006;0001)
-BR [F]
-B [F]
00 e 000BA41E(0000;0001)
01 e 000BA432(0000;0001)
-■
```

Command line set

書式 C <文字列>

機能 コマンドラインの設定

解説 デバッグ対象プログラムに与えるコマンドライン文字列を指定します。
このコマンドは、デバッグ対象プログラムを実行する前に指定してください。

例

```
-C a:b:c:d:e:f Ⓜ
-T Ⓜ
-?argv[0];s Ⓜ      ←プログラムに渡されるコマンドラインの文字列表示
  "G:¥xor.x"        パス名付の実行ファイル名
-?argv[1];s Ⓜ      ←プログラムに渡されるコマンドラインの文字列表示
  "a:b:c:d:e:f"      Cコマンドで設定した文字列
-■
```

Dump memory

書 式 D[<サイズ>] [<レンジ>]

機 能 メモリ内容のダンプ

解 説 メモリ内容を 16 進と ASCII コードでダンプ表示します。
<サイズ>は、ダンプするサイズを指定します。
(S=バイト、W=ワード、L=ロングワード)
<サイズ>の指定を省略すると、W となります。
<レンジ>は、ダンプする範囲を指定します。
<レンジ>を省略した場合は、以前に D コマンドで表示された最後のデータの次から 128 バイト分表示されます。

例

```
-DS BA4E4 BA533 ②          ←バイト単位のダンプ表示
000BA4E4  43 20 6C 69 62 72 61 72 79 20 66 6F 72 20 58 36  C library for X6
000BA4F4  38 30 30 30 20 58 43 BA DD CA DF B2 D7 20 76 32  8000 XCコンパイル v2
000BA504  2E 30 30 00 43 6F 70 79 72 69 67 68 74 20 31 39  .00.Copyright 19
000BA514  38 37 2C 38 38 2C 38 39 2C 39 30 20 53 48 41 52  87,88,89,90 SHAR
000BA524  50 2F 48 75 64 73 6F 6E 00 00 4F F9 00 0B D1 38  P/Hudson..O...48
-D BA4E4 L50 ②          ←ワード単位の長さ指定のダンプ表示
000BA4E4  4320 6C69 6272 6172 7920 666F 7220 5836  C library for X6
000BA4F4  3830 3030 2058 43BA DDCA DFB2 D720 7632  8000 XCコンパイル v2
000BA504  2E30 3000 436F 7079 7269 6768 7420 3139  .00.Copyright 19
000BA514  3837 2C38 382C 3839 2C39 3020 5348 4152  87,88,89,90 SHAR
000BA524  502F 4875 6473 6F6E 0000 4FF9 000B D138  P/Hudson..O...48
-■
```


Examine

書 式 E? <C 評価式> [; <フォーマット>]

機 能 C 評価式の設定

解 説 C 言語の式を設定し、評価値を表示します。
 設定できる評価式は、1) ~9) の9個までです。
 設定された式は、ユーザープログラムから SCD に制御が移るたびに再評価され、表示されます。
 <C 評価式> は、C 言語の式の記述と同じ形式で記述します。
 <フォーマット> は、評価値を表示する書式を指定します。
 この場合の<C 評価式> は、副作用のある演算子(代入など)は画面が再表示されるたびに評価されるため、使用すべきではありません。
 <C 評価式> や<フォーマット> については、本章「4.2 C 言語の式表記」を参照してください。

例

```
-T
-E? i                               ←レジスタスクリーンに評価値を表示
-E? *(argv+1);s                     ←レジスタスクリーンに評価文字列を表示
-X
PC=000BA34C USP=000CD51C SSP=000067F2 SR=8014 X:1 N:0 Z:1 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
1) i = 0
2) *(argv+i);s = "G:¥xor.x"
subq.l #1,$0008(A6)                ;000CD52C(00000001)
-■
```

Examine Erase

書 式 EE <C 評価式番号>

機 能 C 評価式の削除

解 説 E?コマンドで設定された C 言語の評価式を削除します。
評価式には、1) ~9) までの番号がついていますので、削除したい番号を<C 評価式番号>に指定してください。
また、W? (ウォッチポイント) コマンドおよび T? (トレースポイント) コマンドで設定された式も削除できます。
その場合は、<C 評価式番号>のところに、W または T を指定してください。

例

```
-X [F]
PC=000BA34C USP=000CD51C SSP=000067F2 SR=8014 X:1 N:0 Z:1 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
w) c==0x1A = 0
t) i (000CD51C:0004)=00 00 00 00
1) i = 0
subq.l #1,$0008(A6) ;000CD52C(00000001)
-EE1 [F]
-X [F]
PC=000BA34C USP=000CD51C SSP=000067F2 SR=8014 X:1 N:0 Z:1 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
w) c==0x1A = 0
t) i (000CD51C:0004)=00 00 00 00
subq.l #1,$0008(A6) ;000CD52C(00000001)
-EET [F]
-X [F]
PC=000BA34C USP=000CD51C SSP=000067F2 SR=8014 X:1 N:0 Z:1 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
w) c==0x1A = 0
subq.l #1,$0008(A6) ;000CD52C(00000001)
-EEW [F]
-X [F]
PC=000BA34C USP=000CD51C SSP=000067F2 SR=8014 X:1 N:0 Z:1 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
subq.l #1,$0008(A6) ;000CD52C(00000001)
-■
```


Fill memory

書 式 F[<サイズ>] <レンジ> <データ>

機 能 フィルメモリ

解 説 <レンジ>で指定したメモリ内を、<データ>で指定した値で満たします。

<サイズ>には<データ>のサイズを指定します。

(S=バイト、W=ワード、L=ロングワード)

<サイズ>の指定を省略すると W となります。

例

```
-DS C0000 C003F 00
000C0000 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF .....
000C0010 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF .....
000C0020 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF .....
000C0030 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF .....
-FS C0000 C003F FF
-DS C0000 C003F FF
000C0000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
000C0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
000C0020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
000C0030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
-■
```

Go

書 式 G[= <開始アドレス>] [<ブレークポイントのアドレス>]

機 能 デバッグ中のプログラムの実行

解 説 デバッグ対象プログラムに実行権を渡します。

<開始アドレス>を指定した場合は、該当プログラムの開始アドレスから、省略された場合は、プログラムカウンタが示すアドレスから実行され、プログラムが終了するかブレークポイントで停止するまで続きます。

なお、G コマンドで指定されたブレークポイントは、指定時のみ有効で、再度停止させたい場合は、再度設定する必要があります。

また、G コマンドでは実行権をデバッグ対象プログラムに渡してしまうため、実行中には条件つきブレークポイント、およびトレースポイントによる停止は働きません。

例

```
File Edit Watch Exec View Trace Step Options 0F.1
10: #define MAXKEY 64
11:
12: static int keylen[MAXKEY];
13: static int keyoff[MAXKEY];
14:
15: main(int argc, char **argv)
16: {
17:     int c, i;
18:
000EFB14 19: if (--argc > MAXKEY) {
000EFB24 20:     fprintf(stderr, "xor: too many keys\n");
000EFB38 21:     exit(1);
22: }
000EFB46 23: ++argv;
000EFB4A 24: (void)setmode(fileno(stdin), O_BINARY);
000EFB68 25: for (i = 0; i < argc; i++) {
000EFB76 26:     if ((keylen[i] = strlen(argv[i])) == 0)
000EFB9E 27:         keylen[i] = 1;
R=00000000 00 00000000 00 00000000 00 00000000
PC=000EFB14 D0=000EFF54 D4=00000000 A0=000F02CC A4=000EFC08
USP=00102CC6 D1=00000000 D5=00000000 A1=000F0102 A5=000EFA20
SSP=000067F2 D2=00000001 D6=00000000 A2=000F00FA A6=00102CCE
D3=00000000 D7=00000000 A3=0008B1D0 A7=00102CC6
[Commands]
loading XOR.X
-G .19
←19行目にブレークポイントを設定して実行
```


Help

書 式 H

機 能 オンラインヘルプメッセージ

解 説 SCD のコマンドの使い方の簡単な説明を表示します。
表示内容には、コマンド一覧、コマンド表記、C 言語の式表記があります。

例

```

File Find Watch Exec Show Trace Step Option
000749F0 3F3C
000749F4 3F3C
000749F8 6100
000749FC 508F
000749FE 4879
00074A04 3F3C
00074A08 3F3C
00074A0C 6100
00074A10 508F
00074A12 6100
00074A16 3F3C
00074A1A 3F3C
00074A1E 302E
00074A22 C0FC
00074A26 D07C
00074A2A 3F00
00074A2C 3F3C
00074A30 6100
PC=000749F0 D
USP=00047CB2 D
SSP=000067F2 D

SCDのコマンド一覧 1

A[address]           : アセンブル
AN[address]          : アセンブル(ニーモニック表示なし)
B                    : ブレークポイントの表示
B[bp] address [count] : ブレークポイントの設定
BC bp                : ブレークポイントの削除
BD bp                : ブレークポイントの無効化
BE bp                : ブレークポイントの有効化
BR                    : ブレークカウンタの初期化
C string              : コマンドラインの設定
D[size][range]        : メモリ内容のダンプ
E? Cexp[;format]      : C評価式の設定
EE num                : C評価式の削除
F[size]range data     : フィルメモリ
G[=address] [address] : デバッグ中のプログラムの実行
H                      : オンラインヘルプメッセージ
HC                    : トレース履歴の消去
HI [-count]           : トレース履歴の表示
I[..filename]         : ソース表示モードへの切り替え
-- More -- (y/n)? ■
  
```

-H ②

コマンド一覧の一部です。

History Clear

書式 HC

機能 トレース履歴の消去

解説 トレースの履歴を消去します。

例

```
File Edit Watch Breakpoint View Trace Step Option Help
14:
15: void main(int argc, char **argv)
16: {
17:     int    c, i;
18:
00075644 19:     if (--argc > MAXKEY) {
00075654 20:         fprintf(stderr, "xor: too many keys\n");
00075668 21:         exit(1);
-----
PC=00075676 D0=00000000 D4=00000000 A0=00075D02 A4=000757EE
USP=00087E06 D1=00000000 D5=00000000 A1=00075BE4 A5=00075550
SSP=000067F2 D2=00000001 D6=00000000 A2=00075BDC A6=00087E0E
D3=00000000 D7=00000000 A3=000351D0 A7=00087E06
-----
Command:
-
-hi
-- Trace history --
00075652 ble.s $00075676
0007564C cmp.l #$00000040,D0
00075648 move.l $0008(A6),D0
19: if (--argc > MAXKEY) {
00075644 subq.l #1,$0008(A6)
_main
00075640 bra.w $000757A6
_main
000757EE lea $00077A22,A7
-hc
-hi
-- Trace history --
-■
```

History Trace

書 式 HI [-<カウント>]

機 能 トレース履歴の表示

解 説 トレース履歴を新しい順に表示します。

トレース履歴は、T,Uなどのコマンドでプログラムがトレース実行された場合のPC (プログラムカウンタ)を、最近のものから256ステップ分記憶しています。

Gコマンドで実行された場合は、記憶されません。

HIの後ろに-<カウント>(2~ff)を指定すると、<カウント>ステップ前からのヒストリーを古い順に表示します。

例

```
-HI 2
--Trace history--
00100658      movea.l  D0,A0
00100656      addq.l   #4,D0
00100652      move.l   $000C(A6),D0
0010064C      move.l   #$0010263E,-(A7)
33:           sscanf(argv[1],"%d",&precision);
00100646      move.l   #$001074CC._precision,-(A7)
00100644      bne.s    $0010066E
0010063E      cmp.l    #$00000002,D0
32:           if(argc==2) {
0010063A      move.l   $0008(A6),D0
-main 2
00100636      bra.w    $0010078C
-main 2
0010141A      lea      $00107CBA,A7
-■
```


Change Source

書式 I[..<ファイル名>]

機能 ソース表示モードへの切り替え

解説 リストスクリーンの表示をソース表示に切り替えます。

表示ソースファイルを指定する場合は、2つのピリオド(..)の後ろに拡張子を除いたファイル名を指定します。

I のみの場合は、現在のソースを表示します。

なお、リストスクリーンについて詳しくは、本書「第5章 フルスクリーン」を参照してください。

例



```
File Find Watch Exec Show Trace Step Option xor.
17: int c, i;
18:
000EFB14 19: if (--argc > MAXKEY) {
000EFB24 20:     fprintf(stderr, "xor: too many keys\n");
000EFB38 21:     exit(1);
22: }
000EFB46 23: ++argv;
000EFB4A 24: (void)setmode(fileno(stdin), O_BINARY);
000EFB68 25: for (i = 0; i < argc; i++) {
000EFB76 26:     if ((keylen[i] = strlen(argv[i])) == 0)
000EFB9E 27:         keylen[i] = 1;
000EFBB2 28:     keyoff[i] = 0;
29: }
000EFBC8 30: while ((c = getchar()) != EOF) {
000EFBDC 31:     for (i = 0; i < argc; i++) {
000EFBEA 32:         c ^= argv[i][keyoff[i]];
000EFC16 33:         keyoff[i] = (keyoff[i] + 1) % keylen[i];
34:     }

```

PC=000EFC08 D0=00000000 D4=00000000 A0=000EFA10 A4=000EFC08
USP=000C1C70 D1=00000000 D5=00000000 A1=000F28E2 A5=00000000
SSP=000067F2 D2=00000000 D6=00000000 A2=000C2704 A6=00000000
D3=00000000 D7=00000000 A3=0008B1D0 A7=000C1C70

loading xor.x
-I ..xor.17 ←xor.cのソースリスト17行目からリストスクリーンに表示

List Source

書式 I?

機能 ソースファイルの一覧

解説 リストスクリーンの表示をソース表示に切り替え、ソースファイルの一覧を表示します。

例

```

1: #include <stdio.h>
2:
3: int xdot;
4:
5: main()
6: {
7:     xdot = 100;
8:     printf("xdot(main) = %d\n", xdot);
9:     sub();
10: }
11:

```

PC=000EF6A6	D0=00000000	D4=00000000	A0=000EF4F0	A4=000EF6A6
USP=000C1C70	D1=00000000	D5=00000000	A1=000F1AD2	A5=00000000
SSP=000067F2	D2=00000000	D6=00000000	A2=000C2704	A6=00000000
	D3=00000000	D7=00000000	A3=0008B1D0	A7=000C1C70

```

loading main.x
-l?
* 0: main.c ←*は現在実行されているソースファイル
  1: sub.c

```

List

書 式 L [<レンジ>]

機 能 リスト表示 (ソース/アセンブリ)

解 説 指定された<レンジ>内の、デバッグ中のプログラムリストを表示します。

ソース表示モードのときは、C言語ソースリストを、アセンブラ表示モードのときは、逆アセンブルリストを表示します。

<レンジ>には、<開始アドレス>と<終了アドレス>を指定します。

フルスクリーンモードでは、<開始アドレス>のみ有効です。

コンソールモードで<終了アドレス>を省略すると、一部のリスト表示となります。

<開始アドレス>を省略すると、前回のコマンド実行時に表示された最終行の次の行から表示します。

また、ソース表示モードのときには、2つのピリオド (..) の次に拡張子を除いたファイル名を指定することができ、1つのピリオド (.) の次に行番号を指定することができます。

例

```
-L ...xor.1 ②                                     ←xor.cのCソースリストを表示
1: /*
2:  *      xor.c - simple crypt
3:  */
4:
5: #include      <stdlib.h>
6: #include      <stdio.h>
                  ←コンソールモードでの表示例
                  フルスクリーンモードでは
                  リストスクリーンにスクリーン
                  サイズ行分のリスト表示
13: static int n;
14:
15: void main(int argc, char *argv[])
16: {
-0 ②
-L ._main ②
_main
000BA340      link    A6, #FFFF8
17:          int c = 0, i = 0;
000BA344      clr.l   $FFFC(A6)
000BA348      clr.l   $FFF8(A6)
19:          if (--argc > MAXKEY) {
000BA34C      subq.l   #1, $0008(A6)
000BA350      move.l   $0008(A6), D0
000BA354      cmp.l    #$00000040, D0
000BA35A      ble.s    $000BA37E
20:          fprintf(stderr, "xor: too many keys\n");
000BA35C      move.l   #$000BA4D0, -(A7)
-■
```


Memory Edit

書 式 ME[<サイズ>] [<アドレス>] [<データ>...]
MEN[<サイズ>] [<アドレス>]

機 能 メモリ内容の編集

解 説 メモリ内容を編集します。

<サイズ>は、編集するメモリのサイズを指定します。

(S=バイト、W=ワード、L=ロングワード)

<アドレス>は、編集する開始アドレスを指定します。

<データ>は、メモリに入れる内容です。

<データ>を省略すると、現在のメモリ内容をいったん表示して入力モードになります。

入力モードから抜けるには、ピリオド(.)を入力してリターンキー[Enter]を押してください。

MEN コマンドは、ME コマンドと同じ機能ですが、入力モードのときにメモリ内容を表示しません。

<サイズ>に P を指定した場合には、バイトアクセスでアドレスが2増加します。

例

```
-D BA4D0 L20[Enter]
000BA4D0 786F 723A 2074 6F6F 206D 616E 7920 6B65 xor: too many ke
000BA4E0 7973 0A00 4320 6C69 6272 6172 7920 666F ys..C library fo
-ME BA4D0 'tes:'[Enter]
-D BA4D0 L20[Enter]
000BA4D0 7465 733A 2074 6F6F 206D 616E 7920 6B65 tes: too many ke
000BA4E0 7973 0A00 4320 6C69 6272 6172 7920 666F ys..C library fo
-ME BA4D0[Enter]
000BA4D0          7465733A : 'xor:'[Enter]
000BA4D4          20746F6F : .[Enter]

-D BA4D0 L20[Enter]
000BA4D0 786F 723A 2074 6F6F 206D 616E 7920 6B65 xor: too many ke
000BA4E0 7973 0A00 4320 6C69 6272 6172 7920 666F ys..C library fo
-■
```

Memory Move

書 式 MM <レンジ> <アドレス>

機 能 メモリ内容の移動

解 説 <レンジ>で指定するメモリブロックを、<アドレス>で指定した位置へ移動します。

<レンジ>で指定した移動元アドレスのメモリの内容は、本コマンド実行後もそのまま残っています。

ただし、移動元ブロックと移動先ブロックが一部重なった場合には、重なった部分のメモリの内容は変更されます。

例

```
-D BA4E4 L50 ②
000BA4E4 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000BA4F4 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000BA504 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000BA514 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000BA524 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
-MM BA4E4 L50 C0000 ②
-D C0000 L50 ②
000C0000 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000C0010 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000C0020 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000C0030 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000C0040 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
-■
```


Memory Search

書 式 MS[<サイズ>] <レンジ> <データ>

機 能 メモリ内容の検索

解 説 <レンジ>で指定したメモリ内を検索し、<データ>で指定した値を捜します。
 <サイズ>は、<データ>のサイズを指定します。
 (S=バイト、W=ワード、L=ロングワード)
 <サイズ>の指定を省略すると、W となります。
 <データ>は、複数のデータも指定できます。
 シングルクォート(')で囲まれた文字列を指定することもできます。
 その場合は、<サイズ>は自動的に S になります。

例

```
-MS BA340 BD138 4E75 [F]
000BA4CE 000BA8EA 000BA906 000BA90A 000BA940 000BA9F6 000BAA1E 000BAA22
000BAA3A 000BAA60 000BAA6C 000BAA70 000BAACE 000BAB18 000BAC02 000BAC06
000BAC18 000BAC1C 000BAC44 000BAC52 000BAC64 000BAC9C 000BACD2 000BACDA
000BACF0 000BADFC 000BAE0C 000BAE22 000BAE2E 000BAE36 000BAE42 000BAE54
000BAE62 000BAE9E 000BAEAC 000BAEF8 000BAF4C 000BAFAA 000BAFB2 000BB018
000BB126 000BB148 000BB180 000BB19C 000BB1A6 000BB316 000BB45A 000BB4A4
000BB4EA 000BB50C 000BB546 000BB570 000BB57C 000BB7EC 000BB80E 000BB850
000BB858 000BB8BC 000BB9F6 000BBA1A 000BBC8A 000BBC9A 000BBD12 000BBD96
000BBE6C 000BBE80 000BBEBC 000BBED0 000BBEE0 000BBF38 000BBF74 000BC086
000BC08A 000BC0C6 000BC0CA 000BC106 000BC114
-MMS BA340 BD138 'xor:' [F]
000BA4D0
-■
```

Object mode

書 式 0

機 能 アセンブリ表示モードへの切り替え

解 説 ソース表示モードからアセンブリ表示モードに切り替えます。
これによって、表示だけでなくプログラム実行の単位も、ソース1行単位から機械語の1命令単位に変更されます。
ソース表示モードに戻る場合は、I コマンドを使用します。

例

```
19: if (--argc > MAXKEY) {
000EFB14 53AE 0008 subq.l r1, $0008(A6)
000EFB18 202E 0008 move.l $0008(A6), D0
000EFB1C B0BC 0000 0040 cmp.l #$00000040, D0
000EFB22 6F22 ble.s $000EFB46
20: fprintf(stderr, "xor: too many keys\n");
000EFB24 2F3C 000E FC7A move.l #$00EFC7A, -(A7)
000EFB2A 2F3C 000F 1ED0 move.l #$00F1ED0, -(A7)
000EFB30 4EB9 000F 06DA jsr _fprintf
000EFB36 508F addq.l #8, A7
21: exit(1);
000EFB38 2F3C 0000 0001 move.l #$00000001, -(A7)
000EFB3E 4EB9 000F 0164 jsr _exit
000EFB44 508F addq.l #4, A7
23: ++argv;
000EFB46 58AE 000C addq.l #4, $000C(A6)
24: (void)setmode(fileno(stdin), O_BINARY);
000EFB4A 2F3C 0000 0400 move.l #$00000400, -(A7)

PC=000EFB14 D0=000EFF54 D4=00000000 A0=000F02CC A4=000EFC08
USP=00102CC6 D1=00000000 D5=00000000 A1=000F0102 A5=000EFA20
SSP=000067F2 D2=00000001 D6=00000000 A2=000F00FA A6=00102CCE
D3=00000000 D7=00000000 A3=0008B1D0 A7=00102CC6

loading xor.x
-T
-O
-
```


Print status

書式 P

機能 システムステータスの表示

解説 デバッガ、およびデバッグ中のプログラムのアドレスを表示します。

例

```

1: /*
2:  *   xor.c - simple crypt
3:  */
4:
5: #include <stdlib.h>
6: #include <stdio.h>
7: #include <fcntl.h>
8: #include <io.h>
9:
10: #define MAXKEY 64
11:
12: static int keylen[MAXKEY];
13: static int keyoff[MAXKEY];
14:
15: main(int argc, char **argv)

```

```

PC=000EFC08 D0=00000000 D4=00000000 A0=000EFA10 A4=000EFC08
USP=000C1C70 D1=00000000 D5=00000000 A1=000F28E2 A5=00000000
SSP=000067F2 D2=00000000 D6=00000000 A2=000C2704 A6=00000000
D3=00000000 D7=00000000 A3=0008B1D0 A7=000C1C70

```

```

loading xor.x
-P
debug program from $000AEE80 →デバッガの先頭アドレス
user program from $000EFB10 →ユーザープログラムの先頭アドレス
end $000F28E2 →ユーザープログラムの終了アドレス
exec $000EFC08 →ユーザープログラムの実行アドレス
symbol table from $000EE9C0 →シンボルテーブルがない場合は表示されません

```


Print Function

書 式 PF

機 能 関数リストの表示

解 説 現在、読み込まれているソースファイル内の関数の一覧を表示します。

例

```
-PF   
  main(argc,argv)  
-■
```


Print Global variable

書 式 PG

機 能 グローバル変数リストの表示

解 説 ソースファイル内のグローバル変数の一覧を表示します。

例

```
-PG [J]  
keylen=Array:0x000BC46E  
keyoff=Array:0x000BC56E  
errno=0  
errno=0  
_doserrno=0  
environ=0x000CD538  
_PSP=762448  
_fmode=0  
sys_errlist=Array:0x000BC132  
sys_nerr=34  
_iob=Array:0x000BC6F2  
_liobuf=Array:0x000BCB5C  
-■
```

Print Local variable

書 式 PL

機 能 ローカル変数リストの表示

解 説 ソースファイル内のローカル変数の一覧を表示します。

例

```
-PL②  
  argc=1  
  argv=0x000BCEE0  
  c=157245283  
  i=1885430635  
-■
```

Print Symbol

書 式 PS

機 能 シンボルテーブルの表示

解 説 シンボルテーブルに登録されている、全シンボル名の一覧を表示します。

例

```
-PS ②
$00010000: _STACK_SIZE
$00010000: _HEAP_SIZE
$000BA340: _main
$000BA52E: __main
$000BA7CC: __stack_over
$000BA94C: _tzname
$000BA954: _tzstn
$000BA958: _tzdtn
$000BA95C: _daylight
$000BA960: _timezone
$000BA9BA: _exit
$000BA9F8: _atexit
$000BAA24: __CLMOD
$000BAA3C: _clock
$000BAA62: _srand
$000BAA6E: __SRAND
$000BAA72: __exit
$000BAA76: _tzset
$000BAB24: _getml
$000BAB26: _getmem
$000BAB28: _malloc
$000BAC08: _fileno
$000BAC1E: _getenv
-- More -- Y/N ?
-■
```

Search and Print Symbol

書 式 PS <シンボル>

機 能 シンボルの検索表示

解 説 <シンボル>で指定したシンボル名をシンボルテーブルから検索して見つかった場合、その値を表示します。
なお、<シンボル>にはシンボル名の先頭何文字かを指定し、その文字が含まれるシンボル名を検索できます。

例

```
-PS_m ②  
$000BA340: _main  
$000BAB28: _malloc  
-■
```


Print Structured

書式 PT

機能 構造体の表示

解説 ソースファイル内の構造体（共用体含む）の一覧を表示します。
フォーマットは、次のようになります。

```
struct <構造体のタグ名> {
    <先頭からのオフセット> <変数の型> <変数名>
    ...
}
```

例

```
-PT ②
struct .fake0 {
    0x0000 int    quot
    0x0004 int    rem
}
struct .fake1 {
    0x0000 int    quot
    0x0004 int    rem
}
struct _iobuf {
    0x0000 char * _ptr
    0x0004 int    _cnt
    0x0008 char * _base
    0x000C int    _flag
    0x0010 int    _bsize
    0x0014 char   _file
    0x0015 char   _pback
    0x0016 char * _fname
}
-■
```

Print Address

書 式 P? <アドレス>

機 能 アドレス式の計算

解 説 <アドレス>式を評価し、値を表示します。
<アドレス>式は、16進数（そのまま）、10進数（¥で始まる）、2進数（_で始まる）、ラベル（ピリオドを付ける）、行番号（ピリオドを付ける）などの表現と、いくつかの演算子（+、-、*、/、%(剰余)、&(論理積)、|(論理和)、!(論理否定)、^(排他的論理和))を組み合わせることができます。

例

```
-P?100*2 ②  
00000200  
-P?¥256*¥2 ②          ¥ ... ¥  
00000200  
-p? .a6+8 ②  
000CD52C  
-■
```

Quit

書式 Q

機能 SCD の終了

解説 SCD を終了させ、親プロセスに制御を戻します

例

```

A>scd -t xor.x                                     ←scd.xの起動
X68k Source Code Debugger v1.00 Copyright 1990 SHARP/Hudson
loading xor.x
PC=000BA52E USP=0008C4D0 SSP=000067F2 SR=0000 X:0  N:0  Z:0  V:0  C:0
D  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
A  000BA240 000BD138 0008CF64 000791D0  000BA52E 00000000 00000000 0008C4D0
__main:
lea    $000BD138,A7                                ;000BD138(00)
-Q [?]
A>■                                                 ←親プロセスがCOMMAND.Xのとき

```


Read file

書 式 R <ファイル名>[,<アドレス>]

機 能 ファイルを読み込む

解 説 <ファイル名>で指定したファイルを、<アドレス>で指定したアドレスから、メモリに読み込みます。
<アドレス>を指定しない場合、以下のように読み込むファイルの種類により動作が異なります。

- X 形式または R 形式の実行可能ファイル

G コマンド等で実行できるように変換してユーザープログラム領域にロードします。

ユーザープログラム領域は P コマンドを実行した時の "user program from \$?????" という表示で知ることができます。

- Z 形式の実行可能ファイル

ファイルのヘッダに格納されているロードアドレスがユーザープログラム領域内であれば、ロードアドレスよりロードします。

この場合そのまま G コマンド等で実行することができます。

ロードアドレスがユーザープログラム領域外であれば、ファイルの内容をそのままユーザープログラム領域にロードします。

この場合、G コマンド等で実行することはできません。

- 実行できないファイル

ファイルの内容をそのままユーザープログラム領域にロードします。

もちろん、G コマンド等で実行することはできません。

<アドレス>を指定した場合は、指定したファイルの種類に関係なく、ファイルの内容をそのまま加工しないで、指定されたアドレスよりロードします。

この場合、R 形式の実行可能ファイルをロードした時のみ G コマンド等で実行することができます。

OS やデバッガ等が使用しているアドレスを指定してはいけません。

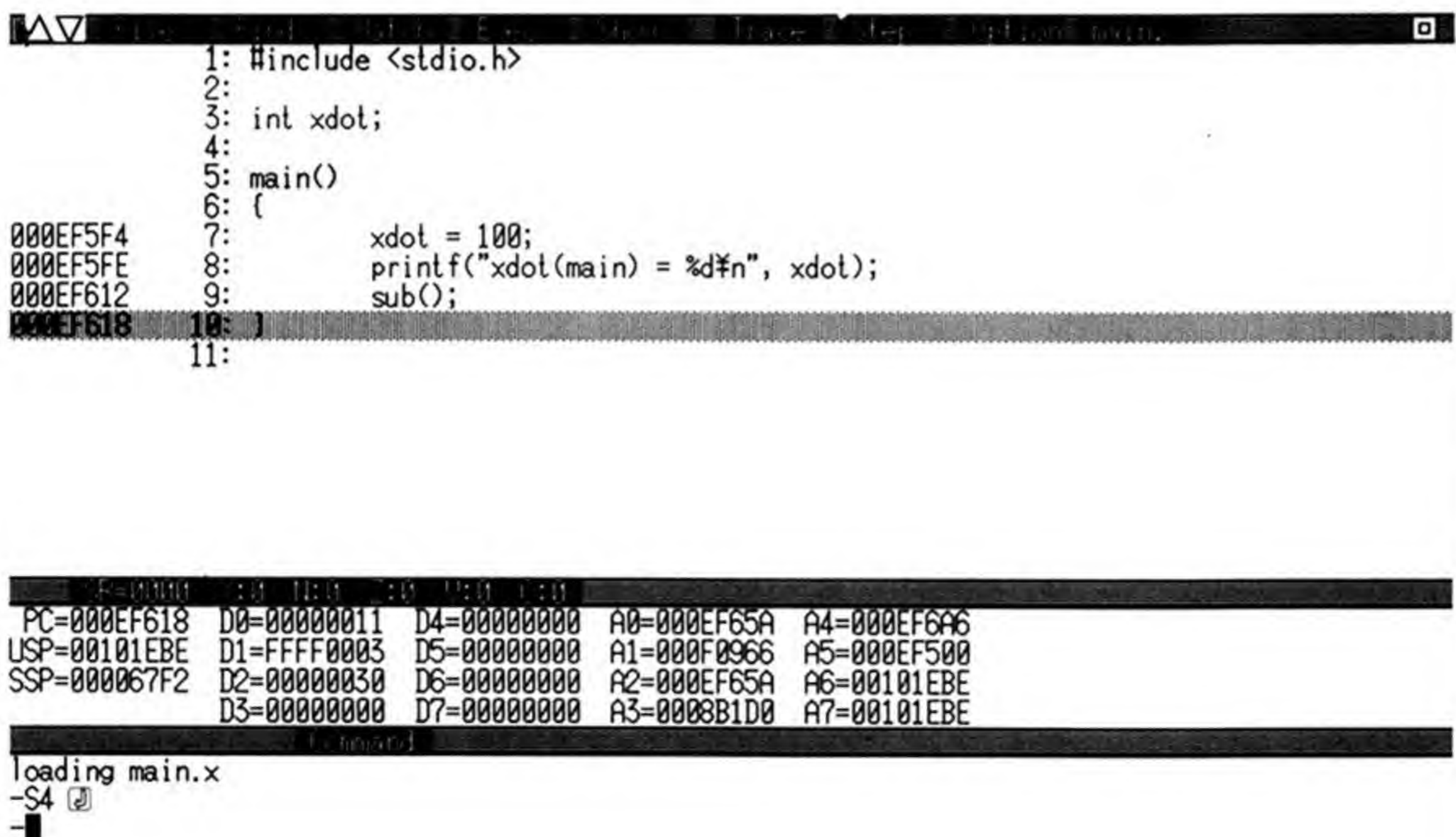
Step

書式 S[<カウント>]

機能 ステップ実行

解説 プログラムを<カウント>の回数分ステップ実行します。
 プログラムカウンタの示すアドレスから実行を開始します。
 もし、<カウント>の回数に到達する前にブレークポイントに出会ったり、条件つきブレークポイントの条件が満たされたり、トレースポイントで指定されているメモリ内容の変化が起きた場合は、そこでステップ操作を中断します。
 ソース表示モードでの1ステップは、実行可能な1行に相当します。
 アセンブリ表示モードでの1ステップは、アセンブル1命令に相当します。
 Tコマンドと違って、JSR (BSR) 命令で呼び出されるサブルーチンは、1命令として実行します。
 ソース表示モードにおいてのステップは、関数呼び出しがその場で実行されるため、呼び出された関数内部をトレースすることはありません。

例



```

1: #include <stdio.h>
2:
3: int xdot;
4:
5: main()
6: {
7:     xdot = 100;
8:     printf("xdot(main) = %d\n", xdot);
9:     sub();
10: }
11:

```

000EF5F4 7: xdot = 100;
 000EF5FE 8: printf("xdot(main) = %d\n", xdot);
 000EF612 9: sub();
 000EF618 10: }
 11:

R=0000	10	11	12	13	14
PC=000EF618	D0=00000011	D4=00000000	A0=000EF65A	A4=000EF6A6	
USP=00101EBE	D1=FFFF0003	D5=00000000	A1=000F0966	A5=000EF500	
SSP=000067F2	D2=00000030	D6=00000000	A2=000EF65A	A6=00101EBE	
	D3=00000000	D7=00000000	A3=0008B1D0	A7=00101EBE	

loading main.x
 -S4
 -

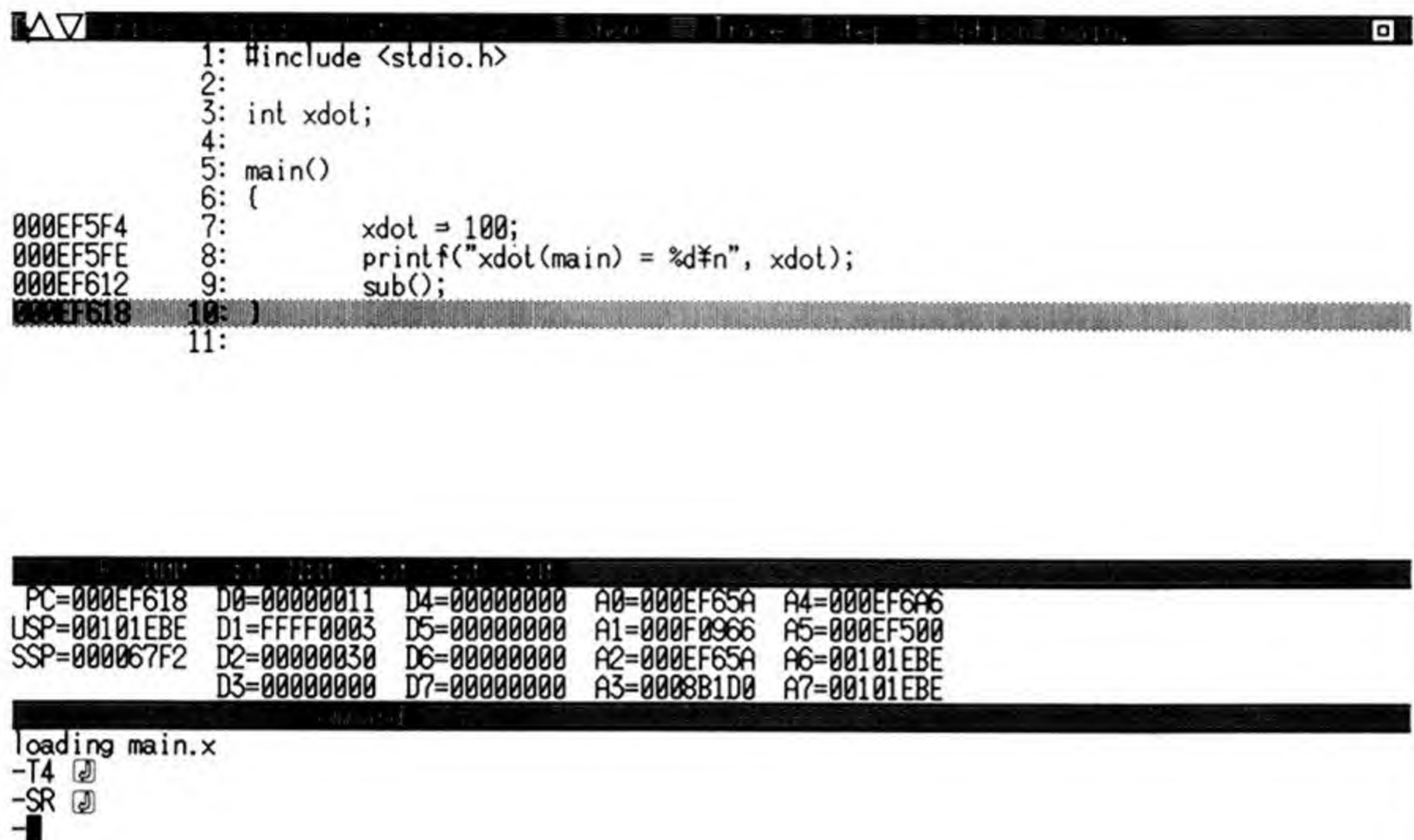
Step until Return

書 式 SR

機 能 関数からの復帰

解 説 RTS, RTR, RTE 命令のいずれかを実行するまで、ステップ実行を続けます。
結果的に、現在の関数（サブルーチン）から復帰することになります。

例



The screenshot shows a debugger window with two panes. The top pane displays C source code with line numbers 1 through 11. Line 10, which contains a closing brace '}', is highlighted in grey. To the left of lines 7 through 10, memory addresses 000EF5F4, 000EF5FE, 000EF612, and 000EF618 are listed. The bottom pane shows a table of register values (PC, USP, SSP, D0-D7, A0-A7) and a command line at the bottom with options -T4, -SR, and a cursor.

```
1: #include <stdio.h>
2:
3: int xdot;
4:
5: main()
6: {
7:     xdot = 100;
8:     printf("xdot(main) = %d\n", xdot);
9:     sub();
10: }
11:
```

PC=000EF618	D0=00000011	D4=00000000	A0=000EF65A	A4=000EF6A6
USP=00101EBE	D1=FFFF0003	D5=00000000	A1=000F0966	A5=000EF500
SSP=000067F2	D2=00000030	D6=00000000	A2=000EF65A	A6=00101EBE
	D3=00000000	D7=00000000	A3=0008B1D0	A7=00101EBE

loading main.x
-T4 ☒
-SR ☒
-

Trace

書式 T[=<アドレス>] [<カウント>]

機能 トレース実行

解説 プログラムを<カウント>回数分トレース実行します。
 <アドレス>を指定した場合には、そのアドレスから、指定しない場合には、プログラムカウンタの示すアドレスから実行を開始します。
 もし、<カウント>の回数に到達する前にブレークポイントに出会ったり、条件つきブレークポイントの条件が満たされたり、トレースポイントで指定されているメモリ内容の変化が起きた場合は、そこでトレース操作を中断します。
 ソース表示モードでの1トレースは、実行可能な1行に相当します。
 アセンブリ表示モードでの1トレースは、アセンブル1命令に相当します。
 Tコマンドでは、JSR (BSR) 命令を実行すると、サブルーチンへ分岐し、サブルーチン内部をさらにトレースで追いかけることができます。
 ソース表示モードにおいてのトレースでは、関数呼び出しが起きたとき呼び出された関数のソースが存在するならば、その関数内部をトレースで追いかけることができます。

例

```

1: sub()
2: {
3:     extern int xdol;
4:
000EF632 5:     printf("xdol(sub) = %d\\n", xdol);
000EF646 6: }
7:

```

PC	D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7
000EF632	00000011	FFFF0003	00000030	00000000	00000000	00000000	00000000	00000000	000EF62C	000F0966	000EF62C	0008B1D0	000EF6A6	000EF500	00101EB6	00101EB6

loading main.x
-T4



Set Tracepoint

書 式 T? <C 評価式> [; <長さ>]

機 能 トレースポイントの設定

解 説 メモリ内容を監視するアドレスと長さを設定します。
指定したアドレス範囲のメモリ内容が変化したときに、プログラムの実行を停止させることができます。
トレース、またはステップ実行時に有効です。
<C 評価式> は、C 言語の式の記述と同じ形式で記述します。
もし、<C 評価式> のところにある式が単純変数だった場合は、そのアドレスとサイズに設定されます。
<長さ> には、監視するメモリのバイト数が指定できます。
<長さ> を省略した場合は、省略値 4 か、もしくはサイズを持つ変数による指定の場合、その変数のサイズに設定されます。
<C 評価式> については、本章「4.2 C 言語の式表記」を参照してください。

例

```
-T? i   
-X   
PC=000BA3BE USP=000CD51C SSP=000067F2 SR=8000 X:0 N:0 Z:0 V:0 C:0  
D 00000800 01C20402 00000400 00000000 00000000 00000000 00000000 00000000  
A 000BCB60 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C  
t) i (000CD51C:0004)=00 00 00 00  
clr.l $FFF8(A6) ;000CD51C(00000000)  
-■
```

Untrace

書式 U[=<アドレス>] [<カウント>]

機能 表示なしトレース

解説 Tコマンドとほぼ同じですが、プログラムの実行が停止するまでトレースの表示は行いません。
すなわち、Tコマンドのように1ステップ実行するたびにトレース表示する、ということはありません。

例

```
-T3
000BA344    17:      int c = 0, i = 0;
000BA34C    19:      if (--argc > MAXKEY) {
000BA37E    23:      ++argv;
-U3
000BA3BE    26:      for (i = 0; i < argc; i++) {
-■
```



Console Change

書 式 V

機 能 コンソールの切り替え

解 説 コマンドの入出力装置を con と aux の間で切り替えます。
aux に切り替えると、デバッガに対する入出力は RS-232C ポートを介して行われるようになります。

例

-V 

Write file

書 式 W <ファイル名>, <レンジ>

機 能 ファイルの書き込み

解 説 メモリの内容をファイルに書き込みます。
<レンジ>で指定したメモリブロックの内容を、<ファイル名>で指定したファイルに書き込みます。

例

```
-D BA4E4 L50
000BA4E4 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000BA4F4 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000BA504 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000BA514 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000BA524 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
-W message,BA4E4 BA52B
-!type message
C library for X68000 XCコンパイル v2.00Copyright 1987,88,89,90 SHARP/Hudson
```

デバッガに戻ります！何かキーを押してください

-■

Set Watchpoint

書 式 W? <C 評価式>

機 能 ウォッチポイントの条件設定

解 説 トレース、またはステップ実行時に停止する条件付きブレークポイントの条件を設定します。

条件（評価式）が真（0 以外の値）になると停止します。

<C 評価式> は、C 言語の式の記述と同じ形式で記述します。

この場合の<C 評価式> は、副作用のある演算子（代入など）は画面が再表示されるたびに評価されるため、使用すべきではありません。

<C 評価式> について詳しくは、本章「4.2 C 言語の式表記」を参照してください。

例

```
-W? c==0x1A
-X
PC=000BA344 USP=000CD51C SSP=000067F2 SR=0010 X:1 N:0 Z:0 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
w) c==0x1A = 0
clr.l $FFFC(A6) ;000CD520(FFFFFFFF)
-■
```


Display Register

書 式 X

機 能 レジスタ内容の表示

解 説 レジスタとフラグの内容をすべて表示します。

例

```
-X
PC=000BA344 USP=000CD51C SSP=000067F2 SR=0010 X:1 N:0 Z:0 V:0 C:0
D 000BA7AA 00000000 00000001 00000000 00000000 00000000 00000000 00000000
A 000BAB22 000BA958 000BA950 000791D0 000BA52E 000BA250 000CD524 000CD51C
w) c=0x1A = 0
t) i (000CD51C:0004)=00 00 00 01
1) i = 1
2) argv[0];s = "G:¥xor.x"
clr.l $FFFC(A6) ;000CD520(FFFFFFFF)
-■
```

Register Change

書 式 X <レジスタ名>

機 能 レジスタ内容の変更

解 説 指定されたレジスタの内容を変更します。
指定可能なくレジスタ名>は以下の通りです。

D0～D7 データレジスタ
A0～A7 アドレスレジスタ
SSP スーパーバイザスタックポインタ
USP ユーザースタックポインタ
SR ステータスレジスタ
CCR コンディションコードレジスタ
PC プログラムカウンタ

レジスタの内容を変更するときは、コマンドとレジスタ名を入力してください。
入力すると、レジスタ名、その現在値、プロンプトが表示されますので、新しい値を入力し、リターンキー ↵ を押してください。

例

```
-XD0  $\text{↵}$   
D0:000BA7AA      1A $\text{↵}$   
-XD0  $\text{↵}$   
D0:0000001A       $\text{↵}$   
-■
```

Yes no ask

書 式 Y/N

機 能 ポーズ

解 説 デバッガのコマンドを一時中断します。
継続する場合には Y を、中止する場合には N をキーインしてください。
なお、リターンキー^②だけ入力されると継続します。

例

```
-Y/N②  
Y/N?
```


Display System Variables

書 式 Z

機 能 システム変数の表示

解 説 全システム変数の現在値を表示します。

システム変数とは、頻繁に使用する値を Z0～Z9 の変数で代用させて、いちいち複雑な値を入力しなくてもテバッグをスムーズに行えるようにしたものです。

例

```
-Z0 BD138  
-Z  
Z0:000BD138  
Z1:00000000  
Z2:00000000  
Z3:00000000  
Z4:00000000  
Z5:00000000  
Z6:00000000  
Z7:00000000  
Z8:00000000  
Z9:00000000  
-■
```

System Variable

書 式 Z<システム変数番号>=<式>

機 能 システム変数の設定

解 説 システム変数の値を設定します。
 <システム変数番号>は、システム変数の番号で、0 から 9 まで指定できます。
 また、システム変数は、. をつけてコマンドラインで参照することができます。

例

```
-Z [F]
Z0:00000000
Z1:00000000
Z2:00000000
Z3:00000000
Z4:00000000.
Z5:00000000
Z6:00000000
Z7:00000000
Z8:00000000
Z9:00000000
-Z0 BD138 [F]
-D .Z0 L30 [F]
000BD138 0023 6606 08C4 0017 60C6 76FF 0C00 0030 .#f..t..`=v....0
000BD148 6606 08C4 001A 6012 0C00 002A 660C 221D f..t..`....*f.".
000BD158 3601 101C 6700 05A6 6018 B03C 0030 6512 6...g..7`.-<.0e.
-■
```

Evaluate expression

書 式 ? <C 評価式> [; <フォーマット>]

機 能 C 言語の式評価

解 説 C 言語の式を評価し、評価値を表示します。

<C 評価式> は、C 言語の式の記述と同じ形式で記述します。

<フォーマット> は、評価値を表示する書式を次のいずれかの文字で指定します。

評価値がアドレスである場合は、16 進数で表示されます。

代入演算子を利用すると、変数の値を変更することができます。

変数のスコープ（可視範囲）は、現在トレース中のプログラムカウンタに依存します。

指定した変数が現在のスコープから見えないときは、参照したり変更することはできません。

<C 評価式> や <フォーマット> については、本章「4. 2 C 言語の式表記」を参照してください。

例

```
-? i 0
-? i=3 3
-? i 3
-? argv[0];s "G:¥xor.x"
-? 3.1416*2 6.2832
-■
```


Output Redirect

書 式 > [<ファイル名>]
>> [<ファイル名>]

機 能 出力リダイレクト

解 説 コンソール出力された本コマンド実行後、ファイル名で指定したファイルにリダイレクトします。

リダイレクトを行っているときも、コンソールには同時に表示されます。

>> コマンドは、ファイル名で指定したファイルがすでに存在している場合に、そのファイルに対して追記書き込みします。

<ファイル名>を省略した場合は、現在行っているリダイレクトを中止します。

例

```
-> outfile.prn②

-D BA4E4 L50
000BA4E4 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000BA4F4 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000BA504 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000BA514 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000BA524 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
->②

-!type outfile.prn②
-
000BA4E4 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000BA4F4 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000BA504 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000BA514 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000BA524 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
-
デバッガに戻ります！何かキーを押してください②
-■
```

Command Logging

書式 >@ [<ファイル名>]

機能 入力コマンドをファイルに保存

解説 本コマンド実行後、コマンド投入スクリーンに投入されるコマンドを、<ファイル名>で指定したファイルに出力することを指定します。
<ファイル名>を省略した場合は、現在行っているコマンドのファイル保存を中止します。
なお、コマンド投入スクリーンについて詳しくは、本書「第5章 フルスクリーン」を参照してください。

例

```
->@ scd.log
-D BA4E4 L50
000BA4E4 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000BA4F4 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000BA504 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000BA514 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000BA524 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
-P
debug program from $000796E0
user program from $000BA340
end $000BD138
exec $000BA52E
symbol table from $000B9220
->@

-!type scd.log
D BA4E4 L50
P

デバッガに戻ります！何かキーを押してください
-■
```


Input Redirect

書 式 < <ファイル名>

機 能 入力リダイレクト

解 説 コマンドの投入を、ファイル名で指定されるファイルから行います。
 ファイルが EOF (エンドオブファイル) に達するまで継続します。
 >@コマンドで出力したファイル名を指定すると (コマンドスクリーンでキーボードから入力した操作に限り)、SCD の実行を再現させることになります。

例

```
-!type scd.log␣
D BA4E4 L50
P

デバッガに戻ります！何かキーを押してください␣

-< scd.log␣

-d ba4e4 L50
000BA4E4 4320 6C69 6272 6172 7920 666F 7220 5836 C library for X6
000BA4F4 3830 3030 2058 43BA DDCA DFB2 D720 7632 8000 XCコンパイル v2
000BA504 2E30 3000 436F 7079 7269 6768 7420 3139 .00.Copyright 19
000BA514 3837 2C38 382C 3839 2C39 3020 5348 4152 87,88,89,90 SHAR
000BA524 502F 4875 6473 6F6E 0000 4FF9 000B D138 P/Hudson..O...48
-P
debug program from $000796E0
user program from $000BA340
end $000BD138
exec $000BA52E
symbol table from $000B9220
-■
```


OS Command

書 式 ![<OS コマンド>]

機 能 OS コマンドの実行

解 説 OS (Human68k) のコマンドを実行します。

環境変数 PATH を検索して PATH に登録されたディレクトリを検索し、実行ファイルを捜し出して実行します。

もし、検索に失敗した場合は、コマンドプロセッサ (COMMAND. X) を呼び出してコマンドプロセッサに実行させます。

SCD が起動時に確保したメモリ上で実行しますので、実行するコマンドにはメモリ容量的な制限を受けます。

また、実行したコマンドがデバッグ対象プログラムやその環境に対して影響を与える可能性も考えられるので、注意しなくてはなりません。

<OS コマンド> を省略した場合は、COMMAND. X が起動します。

COMMAND. X から SCD に戻るときは、EXIT コマンドを使用してください。

例

```
-!dir *.c
```

```
REI                                A:¥
  1 ファイル                      88K Byte 使用中      1133K Byte 使用可能
  ファイル使用量                  1K Byte 使用
xor                                c      729  90-05-05  12:00:00
```

デバッガに戻ります！何かキーを押してください

```
-!
```

```
A>dir *.c
```

```
REI                                A:¥
  1 ファイル                      88K Byte 使用中      1133K Byte 使用可能
  ファイル使用量                  1K Byte 使用
xor                                c      729  90-05-05  12:00:00
```

```
A>exit
```

デバッガに戻ります！何かキーを押してください

```
-■
```

4.4 コマンド一覧表

コマンド名	機 能
A [address]	アセンブル
AN [address]	アセンブル(ニーモニック表示なし)
B	ブレークポイントの表示
B [bp] address [count]	ブレークポイントの設定
BC bp	ブレークポイントの削除
BD bp	ブレークポイントの無効化
BE bp	ブレークポイントの有効化
BR	ブレークカウン트의初期化
C string	コマンドラインの設定
D [size] [range]	メモリ内容のダンプ
E? Cexp [;format]	C 評価式の設定
EE num	C 評価式の削除
F[size] range data	フィルメモリ
G[=address] [adress]	デバッグ中のプログラムの実行
H	オンラインヘルプメッセージ
HC	トレース履歴の消去
HI [-count]	トレース履歴の表示
I[.filename]	ソース表示モードへの切り替え
I?	ソースファイルの一覧
L [range]	リスト表示(ソース/アセンブリ)
ME[size] [address] [data]	メモリ内容の編集
MEN[size] [address]	メモリ内容の編集 (表示なし)
MM range address	メモリ内容の移動
MS[size] range data	メモリ内容の検索
O	アセンブリ表示モードへの切り替え
P	システムステータスの表示
PF	関数リストの表示
PG	グローバル変数リストの表示

4.4 コマンド一覧表

コマンド名	機 能
PL	ローカル変数リストの表示
PS	シンボルテーブルの表示
PS symbol	シンボルの検索表示
PT	構造体の表示
P? address	アドレス式の計算
Q	SCD の終了
R filename [,address]	ファイルの読み込み
S[count]	ステップ実行
SR	関数からの復帰
T[=address] [count]	トレース実行
T? Cexp[;length]	トレースポイントの設定
U[=address] [count]	表示なしトレース
V	コンソールの切り替え (RS-232C)
W filename, range	ファイルの書き込み
W? Cexp	ウォッチポイントの条件設定
X	レジスタ内容の表示
X reg	レジスタ内容の変更
Y/N	ポーズ
Z	システム変数の表示
Z num=exp	システム変数の設定
? Cexp[;format]	C 言語の式評価
> [filename]	出力リダイレクト
>> [filename]	出力リダイレクト (アペンド)
> @ [filename]	入力コマンドをファイルに保存
< filename	入力リダイレクト
![os_command]	OS コマンドの実行

第5章

フルスクリーン

概要

プルダウン

マウス操作

エスケープキー

カーソル

コンソール

プルダウンメニューのコマンド

本章では、マウススクリーン全面を利用して、マウスやカーソルによる視覚的な操作を行うフルスクリーンモードについて、具体的な操作方法を述べながら解説します。

5.1 概要

SCD のフルスクリーンモードは、マウススクリーン全面を使用して、マウスやカーソルによる視覚的な操作を行うモードです。

全画面は横方向に4分割されます。

- 1番上の行には、プルダウンメニューが表示されています。
- 2番目の画面は、C言語のテキストを表示したり、逆アセンブルリストを表示するリストスクリーンです。
- 3番目の画面は、ユーザーレジスタや変数内容を表示するレジスタスクリーンです。
- 4番目の画面は、コマンド投入スクリーンです。
(これはコンソールからの操作に最も近いインターフェースになっています)

通常、カーソルはリストスクリーンかコマンド投入スクリーンのどちらかに存在します。

もし、コマンド投入スクリーンにカーソルが存在するときは、SCDはこれまでのDBと同様に、コマンドをキーボードから入力して、リターンキーを押す方法でコマンドを実行することができます。

[HOME] キーを押すことによって、カーソルの場所を切り替えることができます (リストスクリーン/コマンド投入スクリーン)。

コマンドを実行するより視覚的な方法は、プルダウンメニューをマウスでクリックすることです。

SCDはマウスなしでも操作できるように、ファンクションキーからもプルダウンメニューを選択できます。

[F1] から [F8] までが、プルダウンメニューにそのまま対応しています。

[F9] キーはマウス左ボタンに、[F10] キーは右ボタンにそれぞれ対応しています。

5.2 プルダウン

5.2.1 プルダウンメニューの使い方

フルスクリーンのいちばん上の行にあるプルダウンメニューバーを、マウスでクリックするとメニューがあらわれます。

マウスを下に移動すると、マウスカーソルの位置のメニューが反転表示されます。

選択したいメニューへマウスを移動させたのち、マウスボタンを離すとそのメニューが実行されます。

実行させたくないときは、さらにマウスを移動させてメニューの外でマウスボタンを離してください。

[Trace] および [Step] のプルダウンだけは例外で、クリックしたとたんに実行されます。

もうひとつの選択方法は、ファンクションキーを使用することです。

8つのプルダウンメニューは、それぞれファンクションキー [F1] から [F8] にそのまま対応していますので、引き出したいメニューに対応したファンクションキーを押すことによっても、プルダウンを選択することが可能です。

プルダウンメニューが開いたら、カーソル上下キーでメニューを選択してください。

実行させたいメニューが反転したところでリターンキーを押せば、そのメニューを実行させることができます。

もし実行させたくない場合は、[ESC] キーで中止してください。

カーソル左右キーによって、隣のプルダウンメニューに移ることもできます。

[Trace] および [Step] のプルダウンだけは例外で、[F6] あるいは [F7] キーを押したとたんに実行されます。

メニューバーのいちばん左にある△▽マークをマウスで左クリックすると、リストスクリーンを上下にスクロールさせることができます。

右クリックした場合は、1行ずつスクロールさせることができます。

各コマンドについて詳しくは、本章「5.7 プルダウンメニューのコマンド」を参照してください。

5.2.2 プルダウンメニュー一覧

=△▽[File] [Find] [Watch] [Exec] [Show] [Trace] [Step] [Option]

Load	ファイルの読み込み
Shell	OS コマンドの実行
Exit	SCD の終了

Search	文字列検索
Forward	前方検索
Back	後方検索
Symbol	シンボル検索

Set	評価式の設定
Erase	評価式の解除
WatchPt	ウォッチポイントの設定 (停止条件)
TracePt	トレースポイントの設定 (監視設定)

Run	実行
Slow	低速実行
Restart	再実行
Clear	ブレークポイントの削除
Return	関数からの復帰

Assembly	アセンブリ表示/ソース表示
Register	レジスタ表示のフリップ
Screen	ユーザー画面表示
Stack	スタック内容表示
Calls	関数呼び出し履歴表示

アセンブリ/ミックスモード
シンボル表示フリップ
コード表示フリップ
漢字モード表示フリップ
環境設定モード選択フリップ

Mix
Symbols
Code
Kanji
Custom

5.3 マウス操作

マウスを併用することによって、より視覚的かつスピーディーな操作を行うことができます。

5.3.1 リストスクリーン上での操作

リストスクリーンに表示されたCソーステキスト、あるいは逆アセンブルリストは、画面左上に表示されている△▽マークのクリックによって、表示をスクロールさせることができます。

目的の位置を表示させたところで、リストスクリーンの任意の実行可能な行を左クリックすると、そのアドレスにブレークポイントを設定することができます。

ブレークポイント設定された行には、ブレークポイントのマークの表示が付いて、アンダーライン表示されるようになります。

同じように、リストスクリーンの任意の実行可能な行を右クリックすると、そのアドレスまでプログラムの実行を進めることができます。

(G<アドレス>コマンドと同様)

5.3.2 スクリーンサイズの変更

レジスタスクリーンのタイトルバー、コマンド投入行のタイトルバー、コマンド投入行の最下行のひとつ下の行を左クリックして、マウスボタンを離さずに、マウスを上下に移動させると、各スクリーンの表示行数を変更することができます。

ただし、それぞれのスクリーンには、最少限必要な行数があって、それよりも小さくすることはできないようになっています。

5.4 エスケープキー

[ESC] キーと、それに続く1文字の入力によって行う動作がいくつか定義されています。

これらは、テキストエディタのキー操作に比較的近いものです。

[ESC]・[A] : [HOME] キーと同じく、文字カーソルの位置を切り替えます。

(リストスクリーン/コマンド投入スクリーン)

[ESC]・[Q] : Q コマンドと同じく、SCD を終了します。

[ESC]・[S] : [CLR] キーと同じく、アセンブリ表示モード/ソース表示モードの切り替えを行います。

[ESC]・[W] : 各スクリーンのサイズを変更します。

変更されるスクリーンのタイトルバーが反転表示されますので、カーソル上下キーによってタイトルバーを移動させ、リターンキーで確定させます。

移動させるタイトルバーを切り替えるには、[SPACE] キーを押します。

その他の場合、一般的に [ESC] キーは、選択処理や入力処理を中断させる場合に使用します。

5.5 カーソル

文字カーソルは、リストスクリーンかコマンド投入スクリーンのどちらかに存在します。

SCD 起動直後は、コマンド投入スクリーン上であって、SCD のコマンドを受け付けられる状態にあります。

5.5.1 コマンド投入スクリーン上のカーソル

カーソル左右キーは、入力したコマンドを編集する場合に使用します。

これは、テキストエディタで文字を修正する場合と同じ操作です。

カーソル上キーは、コマンド投入スクリーンを遡って参照したいときに使用します。

遡って参照しているときには、カーソル上下キーでスクロールさせることができます。

参照を中止するときは、カーソル上下以外のキーを押します。

5.5.2 リストスクリーン上のカーソル

カーソルキーで C ソーステキスト、あるいは逆アセンブルリストを参照でき、マウスカーソルのかわりに、オブジェクトのアドレスを指定する役割を果たします。

これによって、マウスを使用しない場合でも、リストを見ながらカーソル位置にブレークポイントを置いたり（[F9] キー）、カーソル位置の命令までプログラムを実行させたり（[F10] キー）することが可能になります。

なお、各ファンクションキーの機能は、このカーソルの位置（コマンド投入スクリーンかリストスクリーン）によって動作が異なります。

詳しくは、巻末「付録 キー操作一覧」を参照してください。

5.6 コンソール

コンソールからコマンドを投入するためには、カーソルは、コマンド投入スクリーンになくてははいけません。

もし、リストスクリーンにある場合は、[HOME] キーでカーソルを移動させてください。

コマンド投入スクリーンでは、コンソールモードと同じように、コマンドを入力できます。

コマンドの実行結果のいくつかは、このスクリーンに表示されます。

表示がスクロールして画面からはみ出てしまった場合は、カーソル上キーで遡って参照することもできます。

スクロールバッファには、最近の 128 行分までの表示が記憶されています。

それより以前の表示は、古い順にバッファから削除されていきます。

以前に入力されたコマンドを呼び戻すには、[ROLL DOWN] キーを使います。

以前に入力されたコマンドは、スクロールバッファに残っていれば、呼び戻すことができます。

5.7 プルダウンメニューのコマンド

File

Load	ファイルの読み込み
Shell	OS コマンドの実行
Exit	SCD の終了

Load

ファイルの読み込み

入力ウィンドウにファイル名を入力してください。

拡張子が .X .R .Z .O .BIN .SYS であれば、バイナリーファイルとして読み込みます。

さらに実行可能であれば、ファイル名の後ろにパラメータを指定することもできます。

拡張子がそれ以外であれば、ソースファイルとして読み込まれます。

ソースファイルとして読み込まれると、リストスクリーンに表示されます。

ソースファイルとしての参照は、一時的なものです。

プログラムを実行したりして、リストスクリーンに他の表示が行われると消去されます。

Shell

OS コマンドの実行

指定した OS のコマンドを実行します。

入力ウィンドウに OS コマンドを入力してください。

何も入力しないでリターンキーを押すと、コマンドプロセッサ（通常は COMMAND. X）が起動します。

SCD に復帰するときは、EXIT コマンドで抜け出します。

Exit

SCD の終了

SCD を終了して OS に戻ります。

Find

Search	文字列検索
Forward	前方検索
Back	後方検索
Symbol	シンボル検索

Search

文字列検索

ソースファイル中の文字列を検索します。

入力ウィンドウに検索したい文字列を入力してください。

現在のカーソルの位置から、指定された文字列を検索し、はじめに見つかった文字列の先頭にカーソルを移動します。

もし、発見できなかった場合は、カーソルは現在位置のままになります。

Forward

前方検索

[Search] で指定した文字列を、もう一度検索します。

[Search] との違いは、文字列の入力が不要であることです。

[Forward] 機能は、キーボードからは [CTRL] + [↑] (アップパーアロー) キーで実行できます。

Back

後方検索

[Search] で指定した文字列を、現在のカーソル位置から遡って検索します。

[Back] 機能は、キーボードからは [CTRL] + [¥] (通貨記号) キーで実行できます。

Symbol

シンボル検索

アセンブラレベルでグローバルシンボルを検索します。

解 説

[Search]、[Forward]、[Back] では、ソースファイル中の文字列を検索します。

したがって、検索後はソースを表示し、カーソルはリストスクリーンに移動し、見つかったソースファイルの位置を指します。

Watch

Set	評価式の設定
Erase	評価式の解除
WatchPt	ウォッチポイントの設定 (停止条件)
TracePt	トレースポイントの設定 (監視設定)

Set

評価式の設定

レジスタウィンドウに表示する評価式を設定します。
入力ウィンドウに設定したい評価式を入力してください。
設定できる評価式は、1)~9)の9個までです。
設定された式は、ユーザープログラムから SCD 制御が移るたびに再評価され、表示されます。

Erase

評価式の解除

[Set]、[WatchPt]、[TracePt] で設定された評価式を削除します。
レジスタスクリーンの評価式には、1)~9) までの番号がついていますので、削除したい番号を入力してください。
また、W? (ウォッチポイント) コマンド、および T? (トレースポイント) コマンドで設定された式を削除できます。
その場合は、番号のかわりに W、または T を入力してください。

WatchPt

ウォッチポイント設定 (停止条件)

条件付きブレークポイントとなる条件式を設定します。
条件式が真 (0 以外の値) になると、プログラムのトレースを停止します。
G コマンドなど、ユーザープログラムに実行権を渡してしまう場合は、この条件式の評価ができないので、停止させることはできません。
T, U, S などのコマンドでは、1 トレースごとに条件式を評価し、停止させることができます。

TracePt

トレースポイント設定 (監視設定)

ウォッチポイントと異なり、メモリ内容が変化したときに、プログラムの実行を停止させるメモリアドレス範囲を指定します。
アドレス指定は、C 言語の式で指定しますが、もし、変数をそのまま指定した場合は、その変数のアドレスを指定したことになります。
その場合は、監視するメモリのバイト数は、変数の占めるサイズと同じです。
アドレスを定数で指定した場合、監視するメモリのバイト数は 4 になります。
バイト数を明示的に指定したい場合は、式の後ろにセミコロン (;) と、サイズを 10 進数で指定します。
監視できるメモリの最大の長さは、4,096 バイトまでです。

Exec

Run	実行
Slow	低速実行
Restart	再実行
Clear	ブレークポイントの削除
Return	関数からの復帰

Run

実行

デバッグ対象プログラムに実行権を渡します。

あらかじめ設定してあるブレークポイントに出会うか、プロセスが終了するまで実行を続けます。

Slow

低速実行

ユーザープログラムを1ステップずつ低速で実行します。

1ステップ実行するごとに、スクリーンをSCD側に切り替えます。

Restart

再実行

ユーザープログラムを再ロードします。

ブレークポイントなどの設定は、失われません。

プロセスが終了してしまったときは、[Restart]によってSCDを抜けることなく、再びデバッグできる状態になります。

Clear

ブレークポイントの削除

設定されているブレークポイントをすべて削除します。

Return

関数からの復帰

RTS, RTR, RTE 命令のいずれかを実行するまで、ステップ実行を続けます。

結果的に、現在の関数（サブルーチン）から復帰することになります。

Show

Assembly	アセンブリ表示／ソース表示
Register	レジスタ表示のフリップ
Screen	ユーザー画面表示
Stack	スタック内容表示
Calls	関数呼出し履歴表示

Assembly

アセンブリ表示／ソース表示

アセンブリ表示／ソース表示のモードを切り替えます。

Register

レジスタ表示のフリップ

レジスタ表示する／しないを切り替えます。

Screen

ユーザー画面表示

ユーザー画面を表示します。

ユーザー画面は、OSの標準出力が表示されているスクリーンです。

SCD画面になっているときは、マウス面が不透明になっているため、見ることはできません。

SCD画面に戻す場合は、どれかキーを押すか、マウスボタンをクリックします。なお、SCD画面の一番下には、ユーザー画面を表示するバーがあるので、そこでマウスをクリックすると、ユーザー画面の下の方だけを常に表示することもできます。

Stack

スタック内容表示

ソース表示モードでは、スタックフレーム（フレームポインタ=A6）の内容を変数名と値で表示します。

アセンブリ表示モードでは、スタックポインタ、あるいはフレームポインタの指しているメモリ領域を16進数でダンプ表示します。

表示範囲の移動は、スタックダンプのウィンドウ左上の△▽マークをクリックするか、カーソル上下キーで可能です。

[A7] をクリックするか、カーソル左を押すとスタックポインタ領域を、[A6] をクリックするか、カーソル右を押すとフレームポインタ領域をそれぞれ選択できます。

Calls

関数呼出し履歴表示

main関数が呼び出されてから現在の関数までの関数呼び出しの履歴を、関数名と実引き数値リストで表示します。

関数表示されている時に関数名をクリックするか、カーソルで反転表示の関数を選択してリターンを押すと、その関数レベルの現在位置をソース表示します。

Trace

解 説

トレース実行を1回だけ行います。

ソース表示モードでは、実行可能な行を1行だけ実行します。

また、ソースの存在する関数については、その関数の内部までトレースします。

アセンブリ表示モードでは、機械語を1命令だけ実行します。

また、JSR 命令や BSR 命令で分岐するサブルーチンの内部までトレースします。

Step

解 説

ステップ実行を1回だけ行います。

ソース表示モードでは、実行可能な行を1行だけ実行します。

また、ソースの存在する関数でも、その関数の内部までトレースしません。

アセンブリ表示モードでは、機械語を1命令だけ実行します。

また、JSR 命令や BSR 命令を1命令として実行し、サブルーチンの内部までトレースしません。

Option

Mix	アセンブリ／ミックスモード
Symbols	シンボル表示フリップ
Code	コード表示フリップ
Kanji	漢字モード表示フリップ
Custom	環境設定モード選択フリップ

Mix

アセンブリ／ミックスモード

アセンブリ表示モードにおけるソース混在表示をするかしないかを選択します。
Mix の前にチェック印が付いているときは、混在モードです。
混在モードでは、当該命令語アドレスに対応するソース行がもし存在すれば、それをアセンブルリスト中に挿入した形で表示します。

Symbols

シンボル表示フリップ

アセンブリ表示モードにおいて、グローバルシンボル名を表示するかしないかを選択します。
Symbols の前にチェック印が付いているときは、シンボル表示モードです。
シンボル表示をしない場合は、命令オペランドはすべて 16 進表示になり、表示中にグローバルラベルは付加されません。

Code

コード表示フリップ

アセンブリ表示モードにおいて、命令コードを 16 進数表示するかしないかを選択します。
Code の前にチェック印が付いているときは、コード表示モードです。

上記オプション選択は、アセンブリ表示モードの表示形式を選択するスイッチになっています。

Kanji

漢字モード表示フリップ

漢字表示モードと英字表示モードを選択するスイッチです。

Custom

環境設定モード選択フリップ

環境設定モードに切り替えるスイッチです。
選択されると、環境設定モードウィンドウがあらわれます。
詳しくは、次ページで紹介します。

Option

環境設定



Save	環境ファイル (SCD. CNF) に環境情報を保存
Load	環境ファイル (SCD. CNF) から環境情報を読み込み
Default	環境情報の初期化
Set	環境情報の設定
Cancel	環境設定モードの中止

PEN COLOR

文字色の選択

△▽マークをマウスでクリックすると、数値が変わります。
 数字の部分をマウスでクリックすると、キー入力で設定できます。
 数値は 0-F (16 進数) です。

BACK COLOR

背景色の選択

△▽マークをマウスでクリックすると、数値が変わります。
 数字の部分をマウスでクリックすると、キー入力で設定できます。
 数値は 0-F (16 進数) です。

PATH

パス名の設定

ソースファイル (.C) を探すディレクトリを指定します。
 [] 内をマウスでクリックし、キー入力で設定します。
 この PATH は、環境ファイル (SCD. CNF) には保存されません。

MEMORY

予約メモリの設定 (OS コマンド実行時に使用)

△▽マークをマウスでクリックすると、数値が変わります。
 数字の部分をマウスでクリックすると、キー入力で設定できます。
 数値は 0 (KB)-2000 (KB) (10 進数) です。

このモードの最中にマウスの右をクリックすると、このモードは終了します。

第 6 章

使 用 例

SCD の実際の使用例

サンプルプログラムの説明

ソースプログラムの作成

ソースプログラムのコンパイル

プログラムの実行

SCD の起動とプログラムのデバッグ

ソースプログラムの修正・再コンパイル・再実行

本章では、C 言語で記述され、XC コンパイラでコンパイルされたプログラムを SCD でデバッグする場合の基本的な手順、およびソースプログラムの作成&コンパイル、プログラムの実行などについて詳述します。

6.1 SCDの実際の使用例

これまでデバッガとして利用されてきた DB は、シンボル情報を参照できましたが、プログラムをアセンブリ言語のレベルでしかデバッグできませんでした。

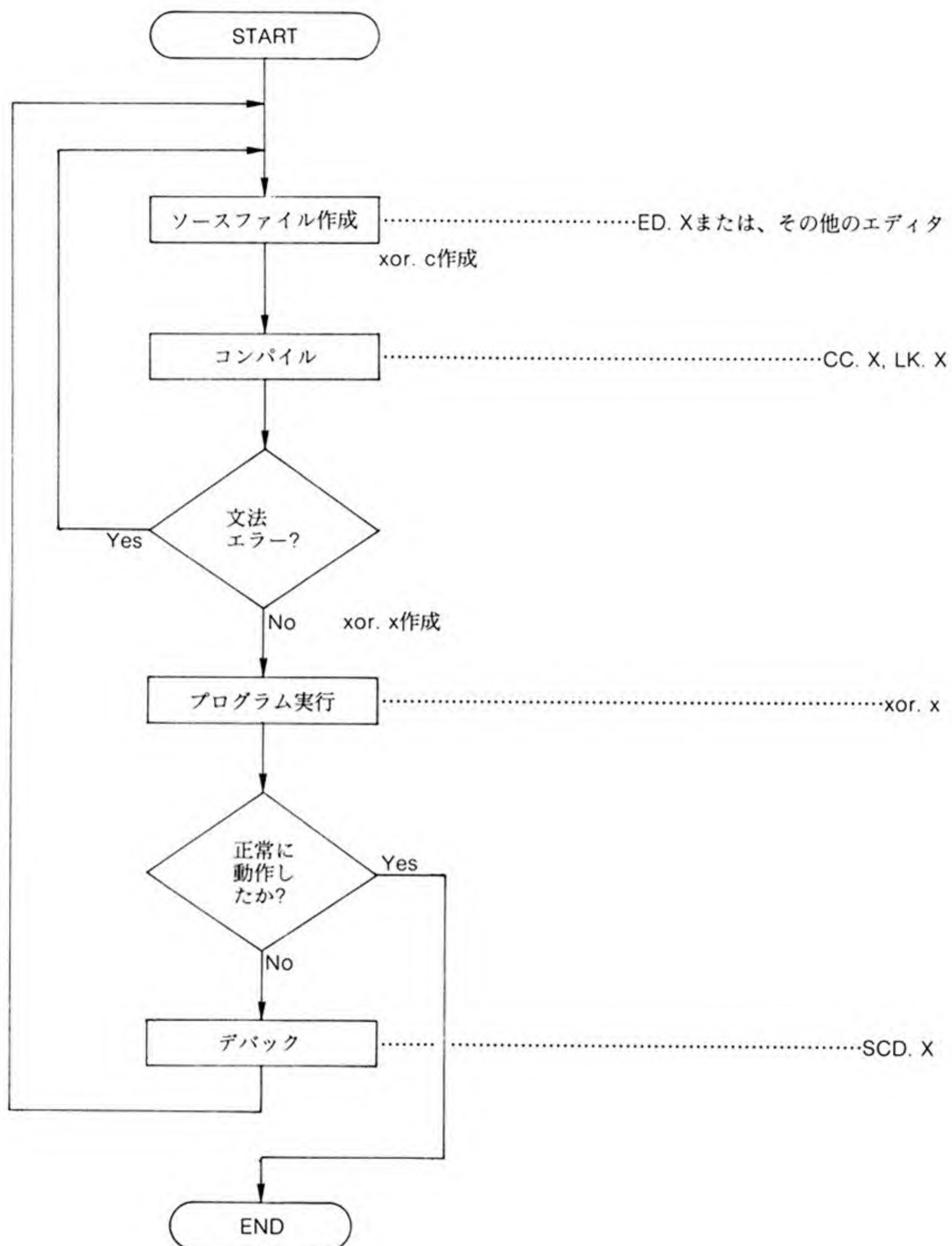
それに対して SCD は、C 言語で記述されたソースプログラムを参照しながら、プログラムをデバッグできます。

つまり、C のソースプログラムの 1 行ごとにステップ実行したり、あるいは C の変数を参照し、その値があらかじめ設定した条件を満たした時に実行を中断する、といったデバッグ方法が可能です。

もちろん、DB のようにアセンブリ言語のレベルのデバッグも可能です。

C 言語で記述され、XC コンパイラでコンパイルされたプログラムを SCD でデバッグする場合、その基本的な手順は次ページのようになります。

6.1 SCDの実際の使用例



左図のようにC言語で記述されたプログラムをデバッグする場合、以下に示すツールやドライバが必要です。

- (1) テキストエディタ (ED. X, etc.)
- (2) Cコンパイラ本体 (CC. X, 但しバージョン 2.00 以上)
- (3) リンカ (LK. X, 但しバージョン 2.00 以上)
- (4) ソースコードデバッガ (SCD. X)
- (5) 浮動小数点演算ドライバ (FLOAT 2. X, FLOAT 3. X)

このうち(5)は、コンパイルする前にメモリ上に常駐させておかなければなりません。

(1)~(4)のツールは、カレントドライブ・カレントディレクトリに関わらず、実行できなければなりません。

そのためには、コマンド検索パス (PATH) を正しく設定する必要があります。

この章では、Aドライブが起動ディスクとランタイムディスク、Bドライブがプログラム格納ディスクとして使用されるものとし、さらに起動ディスク、およびランタイムディスクの構成は、基本的には右の通りであるものと仮定します。

この場合、コマンド検索パスは、最低以下のパス名を含んで設定されるべきです。

path=A:¥;A:¥BIN;A:¥CC;

この他にも、設定すべき環境変数などがありますが、それは後述します。

この章では、SCDによるプログラムのデバッグ方法を、例を挙げながら解説していきます。

起動ディスク

¥

```

|—HUMAN. SYS
|—CONFIG. SYS
|   :
|—AUTOEXEC. BAT
|—COMMAND. X
|—SYS
|   |—FLOAT2. X
|   :   :
|—BIN
|   |—SCD. X
|   |—SCD. CNF
|   |—SCD. HLP
|   |—ED. X
|   |—ED. HLP
|   :   :   :

```

ランタイムディスク

¥

```

|—CC
|   |—CC. X
|—INCLUDE
|   |—ASSERT. H
|   :   :
|—LIB
|   |—CLIB. L
|   :   :
|—BIN
|   |—LK. X
|   |—MAKE. X
|   :   :   :

```

6.2 サンプルプログラムの説明

本章では、以下に示す C のソースリストを使用して解説しています。

```
1: /*
2:  *   xor.c - simple crypt
3:  */
4:
5: #include    <stdlib.h>
6: #include    <stdio.h>
7: #include    <fcntl.h>
8: #include    <io.h>
9:
10: #define     MAXKEY    64
11:
12: static int  keylen[MAXKEY];
13: static int  keyoff[MAXKEY];
14:
15: main(int argc, char **argv)
16: {
17:     int      c, i;
18:
19:     if (--argc > MAXKEY) {
20:         fprintf(stderr, "xor: too many keys\n");
21:         exit(1);
22:     }
23:     ++argv;
24:     (void)setmode(fileno(stdin), O_BINARY);
25:     for (i = 0; i < argc; i++) {
26:         if ((keylen[i] = strlen(argv[i])) == 0)
27:             keylen[i] = 1;
28:         keyoff[i] = 0;
29:     }
30:     while ((c = getchar()) != EOF) {
31:         for (i = 0; i < argc; i++) {
32:             c ^= argv[i][keyoff[i]];
33:             keyoff[i] = (keyoff[i] + 1) % keylen[i];
34:         }
35:         putchar(c);
36:     }
37:     exit(0);
38: }
```


このプログラムは、標準入力から入力される文字と、コマンドラインより引き渡されるパラメータとの排他的論理和 (XOR) をとり、その結果を標準出力に出力する、という働きをします。

以下は、このプログラムの各行における解説です。

5～8 行：必要なヘッダファイルをインクルードします。

10 行：パラメータの最大数をマクロ定義します。

12, 13 行：それぞれ、パラメータの文字列の長さと、その中で参照している文字へのオフセットの値を格納するための領域を確保します。

19～22 行：パラメータの数が MAXKEY で定義された最大数を越えていないかどうかチェックします。

24 行：この setmode 関数は、標準入力をバイナリモードに変更します。

25～29 行：配列 keylen, keyoff の初期化を行います。

30 行：標準入力より Ctrl-Z が入力されるのをチェックします。

32 行：入力された文字とパラメータ中の 1 文字との XOR をとります。

35 行：標準出力に XOR したデータを出力します。

30～36 行がこのプログラムの中核といえます。

標準入力より入力される文字が同じでも、パラメータの数と各パラメータの長さにより、出力されるデータは異なります。

また、標準入出力を使用しているので、リダイレクトが可能です。

サンプルプログラムの動作チェックの時に、出力をリダイレクトしてディスク上のファイルに格納すれば、ツールを用いて、実際にどのようなデータが生成されたかを調べることができます。

6.3 ソースプログラムの作成

前節のサンプルプログラムを、実行可能なオブジェクトに変換するためには、まず、サンプルプログラムをファイルに格納しなければなりません。なぜなら、XC コンパイラは、C のソースリストを格納したソースファイルしか受け付けないからです。

ソースファイルを作成するには、通常、テキストエディタと呼ばれるツールを利用します。

Human68k には、標準添付のテキストエディタとして、ED というフルスクリーンエディタが存在するので、この節では、この ED を例にとって、ソースファイルを作成します。

もちろん、別のテキストエディタを利用しても構いません。

A ドライブに起動ディスク、B ドライブにプログラム格納ディスクをセットし、カレントディレクトリを B:¥とした後で、次のコマンド行を実行します。

```
B>ed xor. c
```

テキストエディタが起動するので、6.2 で示したサンプルプログラムを書いていきます。

エディタの使い方は、[HELP] キーを押すと使用方法が表示されます。

プログラムを入力し終えたら、セーブした後にエディタを終了します。

具体的には [ESC] キーと [E] キーを押すことでセーブ・終了となります。確認のため、xor. c というファイルが作成されているかどうかチェックします。

```
B>type xor. c
```

6.2 で示したサンプルプログラムと違った表示が行われた場合は、再度 ED を起動し、xor. c を修正します。

6.4 ソースプログラムのコンパイル

ソースファイル xor. c をコンパイルする前に、次の条件が満たされているかどうかチェックしてください。

- 環境変数 lib と include が設定されているか

lib=A:¥LIB

include=A:¥INCLUDE

通常、AUTOEXEC. BAT で設定されています。

- FLOAT2. X または FLOAT3. X がメモリ上に常駐しているか

実行すると、新規に常駐したのか、すでに常駐していたのかが分かる。

通常、CONFIG. SYS で設定されています。

- 空きメモリが十分あるか

CC. X 本体とその作業領域、および CC. X から呼び出される、LK. X をロードする領域、またソースプログラムをロードする領域が必要とされる。

これらの条件が満たされていないと、コンパイラが実行されないか、あるいは実行中にエラーとなります。

コンパイル方法は、以下の通りです。

なお、ここで A ドライブのディスクをランタイムディスクに入れ替えます。

```
B>cc /Ns xor. c
```

オプション/Ns は、SCD のために拡張シンボル情報をオブジェクトに付加することを意味します。


このオプションは、全面的な最適化のオプション/O と同時に指定すると、拡張シンボル情報が付加されなくなるので注意してください。

コンパイラが実行されると、画面に CC, LK のタイトルが順次表示されるはずです。

6.4 ソースプログラムのコンパイル

もし、エラーが発生してコンパイルが中断した場合は、ソースファイルや環境変数を再度チェックして、エラーの原因を取り除いてください。

ひとつもエラーが発生せず、LK（リンカ）まで実行した後に終了したなら、ディレクトリ内容を確認してください。

```
B>dir xor. * 
```

xor. c, xor. o, xor. x, がカレントディレクトリに存在していれば、正常にコンパイル・リンクが行われたことを表します。

これで、実行可能ファイル xor. x が作成されました。

6.5 プログラムの実行

では、xor. x を実行してみましょう。
最初は、パラメータなしで実行してみます。

```
B> xor
```

キー入力待ちになるので、適当にキーボードから文字を入力します。
すると、入力した文字がそのままエコーバックされ、画面に表示されます。
最後にリターンキーを押すと 1 行改行し、入力した文字がそのまま変化せずに表示されます。

```
B>
B>xor
abcdefghijklmnopqrstuvwxyz      ←入力
abcdefghijklmnopqrstuvwxyz      ←出力
0123456789
0123456789
dkdfj uerl fkl KJKEJDFIJFIDJ
dkdfj uerl fkl KJKEJDFIJFIDJ
iiiiirrr;::][
iiiiirrr;::][
^Z
B>
```

この理由は、パラメータの数が 0 なので、6.2 のサンプルプログラムの 32 行目が実行されないため、入力がそのまま素通りして出力されるからです。
また、普通、標準入出力はバッファリングされるため、標準入力がある場合、リターンキーが押されるまで、バッファに蓄えられたデータは出力されません。

ですから、1 行ごとにデータが変換されるように見えるのです。
終了するには、[CTRL] キーを押しながら [Z] キーを押した (CTRL+Z) 後で、リターンキーを押します。

次に、パラメータをいくつか設定して実行します。

6.5 プログラムの実行

```
B>xor abc def ghi
```

適当に英字を入力すると、カーソルが移動したり、ブザーが鳴ることがあります。

これは、入力された文字を変換した結果が制御コード (0x00~0x1f) だった場合に起こる現象です。

さて、このプログラム xor は、そのアルゴリズムから入力と出力の文字数が同じであることが期待されます。

これを確かめるためには、xor の出力をディスク上のファイルへリダイレクトし、そのサイズと入力文字数とを比べるという方法があります。

そこで、次のようなコマンド行を実行します。

```
B>xor a:b:c:d:e:f >test
```

すると、test というファイルが作成され、xor の出力が格納されます。

試しに、スペースを 10 回と、CTRL-Z とリターンをキーボードより入力します。

CTRL-Z とリターンは、変換・出力されないはずなので、test のサイズは入力されたスペースの数と同じ 10 バイトと予想されます。

しかし、確認のため下図のように DIR コマンドを実行すると、test のサイズがわずか 2 バイトしかありません。

Human68k のシステムディスクに納められている DUMP コマンドで test の内容をのぞいてみても、やはり 2 バイトしか格納されていません。

```
B>
B>xor a:b:c:d:e:f > test
B>dir test
ボリュームがありません B:¥
1 ファイル 17K Byte 使用中 1204K Byte 使用可能
ファイル使用量 1K Byte 使用
test 2 90-02-13 22:02:50
B>dump test
00000000 41 1A
B>
```

↑
入力10バイトなのに
出力は2バイトしかない。

A.

サンプルプログラムの xor は、どうもおかしな動作をする場合があることが判明したので、次の節では、SCD を用いてその原因を解明します。

6.6 SCDの起動とプログラムのデバッグ

サンプルプログラム xor を SCD でデバッグする場合、SCD の起動方法は以下のようになります。

なお、ここで A ドライブのディスクを起動ディスクに入れ替えます。

```
B>scd xor. x a:b:c:d:e:f
```

この場合、特にオプションを指定する必要はないので、第1パラメータはデバッグするプログラムのファイル名である xor. x とします。

また、第2パラメータ以降は、xor が実行されるときに渡されるコマンドラインです。

注意すべきことは、SCD を起動するとき、カレントディレクトリにソースプログラムとオブジェクトプログラムが存在しなければならない、ということです。

●初期画面

SCD を起動すると、以下のような初期画面が表示されます。

オプションを指定せずに起動すると、フルスクリーンモードになります。

```
File Edit View Options Window Trace Step Help
1: /*
2:  *   xor.c - simple crypt
3:  */
4:
5: #include <stdlib.h>
6: #include <stdio.h>
7: #include <fcntl.h>
8: #include <io.h>
9:
10: #define MAXKEY 64
11:
12: static int keylen[MAXKEY];
13: static int keyoff[MAXKEY];
14:
15: main(int argc, char **argv)
16: {
17:     int c = 0, i = 0;
18:
000F0AB4 PC=000F0C80 D0=00000000 D4=00000000 A0=000F09B0 A4=000F0C80
USP=000C2C70 D1=00000000 D5=00000000 A1=000F388A A5=00000000
SSP=000067F2 D2=00000000 D6=00000000 A2=000C3704 A6=00000000
D3=00000000 D7=00000000 A3=0008B1D0 A7=000C2C70
loading xor.x a:b:c:d:e:f:
-
```

6.6 SCDの起動とプログラムのデバッグ

一番上の1行は、プルダウンメニューバーと呼ばれ、この部分をマウスでクリックすることにより、ファイルアクセスや検索、変数の監視、プログラムの実行・制御、表示の切り換え、トレース実行、ステップ実行、オプションなどの機能を選択します。

メニューバーのすぐ下のウィンドウは、リストスクリーンと呼ばれ、C言語またはアセンブリ言語でリストを表示します。

リストスクリーンの下のウィンドウは、レジスタスクリーンと呼ばれ、MPU68000のレジスタやCの変数表示などに利用されます。

一番下のウィンドウは、コマンドスクリーンと呼ばれ、キーボードから入力されるコマンドやその結果は、このスクリーンに表示されます。

●コマンドラインの設定

コマンドラインを設定するには、Cコマンドを使用します。

コマンドスクリーンにおいて、以下のコマンドを実行します。

```
-C a:b:c:d:e:f
```

デバッグするプログラムを実行する前に設定します。

前述したように、SCD起動時にも指定できますが、新たにこのコマンドで設定し直すことができます。

●Cの評価式の登録と表示

評価式の登録には、E?コマンドを使用します。

デバッグのために、xorで使用している変数をいくつか登録します。

コマンドスクリーンにカーソルがある時に、以下のようにコマンドをキーボードより入力してください。

```
-E? c;x  
-E? i  
-E? argc  
-E? argv[i];s  
-E? argv[i][keyoff[i]]  
-E? i>10
```


すると、レジタスクリーンに各評価式とその値が表示されます。

The screenshot shows the SCD debugger interface. At the top, there's a menu bar with options like File, Edit, View, Break, Trace, Step, Options, and Help. Below the menu bar, the assembly code is displayed with addresses and instructions. The registers section shows the current values of various registers. The watch window at the bottom displays the values of variables being monitored.

```

000F0A84 17:      int    c = 0, i = 0;
000F0A8C 18:
000F0A8C 19:      if (argc > MAXKEY) {
000F0ACC 20:          fprintf(stderr, "xor: too many keys\n");
000F0AE0 21:          exit(1);
000F0AE0 22:      }
000F0AEE 23:      ++argv;
000F0AF2 24:      (void)setmode(fileno(stdin), O_BINARY);
000F0B10 25:      for (i = 0; i < argc; i++) {
000F0B10 26:          cix = 0;
000F0B10 27:          i = 0;
000F0B10 28:          argc = 2;
000F0B10 29:          argv[i]:s = "B: xor xor.x";
000F0B10 30:          argv[i][keyoff[i]] = 66;
000F0B10 31:          i>10 = 0;
000F0B10 32:      }
000F0B10 33:      loading xor.x a:b:c:d:e:f:
000F0B10 34:      -E? cix
000F0B10 35:      -E? i
000F0B10 36:      -E? argc
000F0B10 37:      -E? argv[i]:s
000F0B10 38:      -E? argv[i][keyoff[i]]
000F0B10 39:      -E? i>10
000F0B10 40:      -l
000F0B10 41:      -
  
```

この例では、変数 `c` は 16 進数で、また `argv[i]` は文字列で表示されます。C 言語の型及び演算子をほとんどサポートしているので、もっと複雑な評価式の値も表示できます。

また、評価式の値は、デバッグしているプログラムから SCD に制御が戻るたびに再評価され、表示されます。

メニューバーを用いても同じことができます。

最初にメニューバーの [Watch] を選択し、次にプルダウンメニューの中から [Set] を選びます。

すると、評価式の入力待ちとなるので、キーボードから式を入力します。

メニューの選択はマウスとファンクションキーのどちらかで行えます。

操作の詳しい方法は、本書「第 5 章 フルスクリーン」の概要を参照してください。

●プログラムの低速実行

登録した評価式の値がどのように変化するかを見るために、プログラムをゆっくり実行することができます。

まず、メニューバーの [Exec] を選択し、そのプルダウンメニューの [Slow] を選ぶことで低速実行が行われます。

この低速実行は、1ステップ実行するたびに画面表示をSCDに切り替え、評価式の再評価・表示などが行われます。

ここでいう1ステップとは、リストスクリーンに表示されているリストの1行が実行されることを指します。

つまり、C言語のリストが表示されているときは、実行可能な行が1行ごとに実行されます。

アセンブリ言語のリストが表示されている場合は、機械語が1命令ごとに実行されます。

このように、SCD上でxorを実行していくと、E?で登録した評価式の値が未定義から初期値、そして、また違う値へと変化していくのがよくわかります。

●ステップ実行とトレース実行

ステップ実行では、関数呼び出しやサブルーチンコールの際に、その関数やサブルーチンの内部までトレースします。

それに対してトレース実行は、関数やサブルーチン内部まではトレースしません。

したがって、関数の内部にバグが潜んでいるかもしれない場合は、トレース実行を、バグのいる箇所がある程度判明して関数内部を調べる必要のない場合は、ステップ実行をそれぞれ選択することで、デバッグの効率が向上します。

さて、xorの場合、ユーザ定義関数はmainのみなので、main以外の関数をトレースする必要はないので、ステップ実行だけを利用することにします（ソース表示モードでは、ソースの存在する関数のみトレース実行できるので、この場合は、トレース実行でも同じ結果が得られる）。

ステップ実行には、Sコマンドを使用します。

-S 10 [F]

この例では、10行分のステップ実行を行います。

なお、トレース実行の場合には、SのかわりにTコマンドを使います。

ステップ実行やトレース実行は、メニューバーを使用しても行えます。

メニューバーの[Step]や[Trace]をマウスでクリックするだけです。

さて、xor.cの30～36行の部分をステップ実行して、先ほど登録した評価式を見ていると、スペースを入力するとputchar関数で0x1aが出力される

ことが確認できます。

そして、この 0x1a はどうやら画面に出力されないようです。

この現象の原因は、実は標準出力をバイナリモードにしていなかったためです。xor. c の 24 行目で標準入力をバイナリモードに変更していますが、標準出力もバイナリモードにしないと、一部のキャラクタ (0x1a など) が画面に現れないのです。

●ウォッチポイントとトレースポイントの設定

xor のデバッグには使用しませんでした。使用頻度の高いコマンドがこのウォッチポイントとトレースポイントです。

ウォッチポイントとは、トレースまたはステップ実行時に、条件 (評価式) が真 (0 以外の値) になると停止する、条件付ブレークポイントのことです。

例えば、次のようにウォッチポイントを設定したとします。

```
-W? c==0x1a [F]
```

この場合、トレース実行やステップ実行の最中に、変数 c が 0x1a に等しくなった時、プログラムの実行が停止します。

トレースポイントとは、メモリの内容が変更された時に実行を停止するよう、その内容を監視するメモリへのポインタのことです。

例えば、次のようにトレースポイントを設定したとします。

```
-T? &keyoff[2] [F]
```

この場合、トレース実行やステップ実行の最中に keyoff[2] の内容が変更された時、プログラムの実行が停止します。

6.7

ソースプログラムの修正・再コンパイル・再実行

前節で xor のバグの原因が分かったので、ソースプログラムを ED で修正します。

手順は「6.3 ソースプログラムの作成」と同様です。

xor. c のソースプログラムの 24 行目と 25 行目の間に、次の 1 行を挿入します。

```
(void) setmode (fileno (stdout), O_BINARY);
```

修正が終わったら、再コンパイルです。

A ドライブのディスクをランタイムディスクに入れ替えてください。

手順は「6.4 ソースプログラムのコンパイル」と同様に、

```
B>cc /Ns xor. c
```

と実行してください。

次に動作確認のため、SCD を起動し、0x1a が画面に出力されるかどうかをチェックします。

A ドライブのディスクを起動ディスクに入れ替えてください。

次のコマンド行を実行します。

```
B>scd xor. x a:b:c:d:e:f
```

SCD が起動したら、次のようにコマンドを実行します。

```
-E? c; x  
-W? c==0x1a
```


これにより、変数 `c` の値が表示され、更に `c` の値が `0x1a` に等しくなったときに、トレースまたはステップ実行が停止されるよう設定しました。

この後、低速実行を何回か行い、ウォッチポイントで停止するのを待ちます。

停止したらステップ実行を行い、修正した `xor.c` の 36 行目の `putchar` で `0x1a` が出力されるのを確認します。

`0x1a` が画面に出力されると、画面がクリアされます。

これは Human68k のコンソールドライバが `0x1a` を画面をクリアする制御コードとして受け取ったことを表します。

つまり、`0x1a` が正常に出力されたことを表します。

これで入力と出力が 1 対 1 に対応するようになりました。

これで `xor` のデバッグは終了です。

あとは `/Ns` オプションをつけずに、`xor.c` を再コンパイルすれば、オブジェクトサイズが小さくなります。

付 録

キー操作一覧
SCD エラーメッセージ一覧

キー操作一覧

ファンクションキー

ファンクションキー	機 能
[F1]	[File] メニューの選択
[F2]	[Find] メニューの選択
[F3]	[Watch] メニューの選択
[F4]	[Exec] メニューの選択
[F5]	[Show] メニューの選択
[F6]	[Trace] メニューの選択
[F7]	[Step] メニューの選択
[F8]	[Option] メニューの選択
[F9]	ブレークポイント設定 = マウスの左クリック機能と同じ
[F10]	指定行までの実行 = マウスの右クリック機能と同じ
[HELP]	オンラインヘルプメッセージの表示
[CLR]	アセンブリ表示モード/ソース表示モードの切り替え
[HOME]	文字カーソルの位置 (リストスクリーン/コマンド投入スクリーン) の切り替え

以下はカーソルの位置によって動作が異なる。

機能 ファンクションキー	リストスクリーン	コマンド投入スクリーン
[ROLL UP]	リストのスクロールアップ動作	コマンド履歴呼び出し
[ROLL DOWN]	リストのスクロールダウン動作	コマンド履歴呼び出し
[↑]	カーソルアップ動作	コマンド投入スクリーンを逆方向スクロールさせる
[↓]	カーソルダウン動作	コマンド投入スクリーンを正方向スクロールさせる (最下行以外)
[←]	カーソル左移動 (ソース表示モードのみ)	カーソル左移動 (投入コマンドの編集)
[→]	カーソル右移動 (ソース表示モードのみ)	カーソル右移動 (投入コマンドの編集)
[↵]	コマンド投入スクリーンに移動	コマンドの投入完了動作
[BS]	無効	カーソルの直前の1文字削除

[ESC] キーを伴う動作

ファンクションキー	機 能
[ESC]・[A]	[HOME] キーと同じく文字カーソルの位置の切り替え (リストスクリーン/コマンド投入スクリーン)
[ESC]・[Q]	Q コマンドと同じく SCD を終了
[ESC]・[S]	[CLR] キーと同じくアセンブリ表示モード/ソース表示モードの切り替え
[ESC]・[W]	各スクリーンサイズの変更 カーソル上下キーによってタイトルバーを移動させ、リターンキーで確定 [SPACE] キーで移動させるタイトルバーを切り替え

SCDエラーメッセージ一覧

エラーメッセージ	意 味
File create error	ファイルが作れない
File read error	ファイルが読み込めない
No symbol file	シンボルファイルがない
Device error	デバイス・エラー
Device write error	デバイス書き込みエラー
offset overs	オフセットが範囲外である
Bad breakpoint number(0-1f)	ブレークポイントの番号が範囲 (0-1f) 外である
break pointer over	すでにブレークポイントがいっぱいである
size over	サイズが大きすぎる
Bad parameter	パラメーターに誤りがある
Undefined symbol	未定義シンボル
Can't make symbol table	シンボルテーブルが作成できない
no break pointer	ブレークポイントがない
Breakpoint list or '*' expected	ブレークポイントリストまたは '*' を指定していない
data too long	データが長すぎる
Command error	コマンドエラー
No Symbol table	シンボルテーブルがない
Expression error	式表現に誤りがある
no process	プロセスは終了済です
Exceptional Abort By bus error By Memory Access of <address>	バスエラー (<address> は、アクセスしようとしたアドレス)
Exceptional Abort By address error By Memory Access of <address>	アドレスエラー (<address> は、アクセスしようとしたアドレス)
fatal error	致命的エラー
Exceptional Abort By underfined instruction at <address>	未定義命令を実行した (<address> は、実行しようとしたアドレス)

エラーメッセージ	意 味
zero divide chk instruction trapv instruction privilege violation double exception in system status display	0 で除算した chk 命令を実行した trapv 命令を実行した 特権違反をした 例外処理中にエラーが発生した
?文字定数オーバーフロー ?式表現の誤り ?文字の誤り ?数字表現の誤り ?文字列の終端記号がない ?式評価スタックあふれ ?変数未定義 ?演算子の誤り ?ポインタの誤り ?メンバー未定義	

索引 50 音順

ア

アセンブリ表示／ソース表示	81
アセンブリ表示モード	14、90
アセンブリ／ミックスモード	81
アドレス	19

イ

インクリメント	21
---------	----

ウ

ウォッチポイント	88
ウォッチポイントとトレースポイントの設定	111
ウォッチポイントの設定	81

エ

エスケープキー	83
---------	----

オ

オブティマイズ	3
オンラインヘルプ	10

カ

拡張シンボル情報	15
カーソル	84
カレントディレクトリ	4
漢字モード表示フリップ	81
関数からの復帰	81
関数呼び出し履歴表示	81
間接演算子	21
環境設定モード選択フリップ	81

キ

キャスト演算子	21
---------	----

ク

グローバルシンボル	87
-----------	----

コ

後方検索	81
コード表示フリップ	81
コマンド投入スクリーン	79、84、85、117
コマンドプロセッサ	86
コマンド名	23
コマンドライン	7、9
コマンドラインの設定	108
コンソール	85
コンソールモード	13
コンフィギュレーションファイル	9
コンパイル	3

サ

再実行	81
サイズ	19
3項演算子	21
サンプルプログラムの説明	100

シ

式	19
システムディスク	99
実行	81
実行ファイル名	7、9
初期画面	107

50 音順

シンボル検索	81
シンボル情報	15
シンボル表示フリップ	81
シンボル名	23

ス

スクロールバッファ	85
スタック内容表示	81
ステップ実行	91
ステップ実行とトレース実行	110
前方検索	81

ソ

ソース表示モード	14、90
ソースファイル	86
ソースプログラムの作成	102
ソースプログラムのコンパイル	103
ソースプログラムの修正・再コンパイル・再実行	112

チ

チャイルドプロセス	4、8
-----------	-----

テ

低速実行	81
テキストエディタ	9、99
デクリメント	21

ト

トレース実行	91
トレースポイント	88
トレースポイントの設定	81

ハ

背景色の選択	93
バイナリーファイル	86
パス名の選択	93
パラメータ	23
評価式の解除	81
評価式の設定	81

フ

ファイルの読み込み	81、86
ファイル名	23
ファンクションキー	117
フォーマット	22
浮動小数点演算ドライバ	4、99
フルスクリーンモード	13、79
プルダウンメニュー	80、86
ブレークポイント	89
ブレークポイントの削除	81
プログラムカウンタ	21
プログラムの実行	105
プログラムの低速実行	109
ヘルプファイル	10

マ

マウススクリーン	79
----------	----

モ

文字色の選択	93
文字列検索	81

ユ

ユーザー画面表示81

ヨ

子約メモリの設定93

ラ

ランタイムディスク99

リ

リストスクリーン79、82、84、85、117

リンカ99

レ

レジスタウィンドウ88

レジスタスクリーン82、88

レジスタ表示のフリップ81

レンジ20

アルファベット順

A

Assemble	24
Assembly	90

B

Back	87
BACK COLOR	93
Break count Reset	30
/b <パレットコード>	8

C

C 評価式	21
Calls	90
Cancel	93
Change Source.....	40
Clear	89
Clear Breakpoint	27
Code	92
Command line set	31
Command Logging.....	72
Console Change	62
Custom	92
/c <メモリサイズ>	8
C の評価式の登録と表示	108

D

Default	93
Disable Breakpoint	28
Display Breakpoint	25
Display Register	65
Display System Variables	68
Dump memory.....	32

E

Enable Breakpoint	29
Erase	88
Evaluate expression	70
Examine	33
Examine Erase	34
Exec	89
Exit	86
[ESC] キーを伴う動作	118

F

File	86
Fill memory	35
Find	87
FLOAT 2.X	4
Forward	87

G

Go	36
----------	----

H

Help	37
History Clear	38
History Trace	39

I

Input Redirect	73
----------------------	----

K

Kanji	92
-------------	----

アルファベット順

L

List	42
List Source	41
Load	86、93

M

MEMORY.....	93
Memory Edit	43
Memory Move.....	44
Memory Search	45
Mix.....	92

O

Object mode.....	46
Option	92
OS Command	74
OS コマンドの実行	81
Output Redirect	71

P

PATH	93
PEN COLOR	93
Print Address	54
Print Function.....	48
Print Global variable.....	49
Print Local variable	50
Print status	47
Print Structured	53
Print Symbol	51
/p <パレットコード>	8

Q

Quit.....	55
-----------	----

R

Read file	56
Register.....	90
Register Change	66
Restart	89
Return	89
Run.....	89
/r.....	8

S

Save	93
SCD のコマンド表記	19
SCD の終了	81
SCD の動作環境	4
Screen	90
Search	87
Search and Print Symbol.....	52
Set	88
Set Breakpoint	26
Set Tracepoint	60
Set Watchpoint	64
Shell	86
Show	90
Slow	89
Stack	90
Step	57、91
Step until Return	58
Symbol	87
Symbols.....	92
System Variable	69

T

Trace	59、 91
TracePt	88
/t	8

U

Untrace	61
---------------	----

W

Watch	88
WatchPt	88
Write file	63

Y

Yes no ask	67
------------------	----

パイオニア株式会社

本社 〒545 大阪市阿倍野区長池町22番22号

電子機器事業本部 〒329-21 栃木県矢板市早川町174番地

液晶映像システム事業部 第2商品企画部

お問い合わせ先 〒162 東京都新宿区市谷八幡町8番地 電話 (03)3260-1161(大代表)

東京支社内 液晶映像システム事業部 第2商品企画部 ソフトウェア担当